MEMORIA TFG

HERRAMIENTA DE GENERACIÓN DE LABERINTOS 3D PARA FABRICACIÓN DIGITAL

Diana González Mina

Grado En Ingeniería Informática

Universidad Pública de Navarra

Este documento es la memoria correspondiente al trabajo de fin de grado realizado por Diana González Mina. En esta primera página se muestra la información general a cerca de dicho documento.

Título: Herramienta de generación de laberintos 3D para fabricación digital

Objetivo: Creación de un generador de laberintos en tres dimensiones en un formato imprimible para poder ser impresos con una impresora 3D.

Estudiante: Diana González Mina

Tutor: Dr. Óscar Ardaiz Villanueva

Lugar: Universidad Pública de Navarra

Titulación: Grado en Ingeniería Informática

Semestre: 8

Resumen

Se desea crear una herramienta que permita a los usuarios crear un laberinto aleatorio en tres dimensiones. La herramienta le permitirá al usuario tanto generar el archivo imprimible para una impresora 3D a partir del laberinto generado, como mover una bola en la pantalla para intentar encontrar la solución.

A pesar de la existencia de herramientas con elementos similares, no hay ninguna que combine todos los elementos que forman la herramienta a crear. En concreto, proveen la generación aleatoria del laberinto, la posibilidad de imprimir el objeto, la posibilidad de obtener un laberinto tridimensional y la posibilidad de que el usuario realice los movimientos de la bola en el laberinto sin necesidad de imprimirlo. Sin embargo, ninguna herramienta combina todos esos elementos.

Para el desarrollo, se utilizarán componentes ya creados y las herramientas existentes serán tenidas en cuenta para tomar decisiones sobre los diferentes aspectos de la herramienta a crear.

Contenido

Introducción

En este trabajo se ha desarrollado una aplicación web que genera laberintos tridimensionales aleatorios. En ella el usuario puede seleccionar el valor de cada dimensión del cubo que contendrá el laberinto y una vez generado, el usuario podrá mover una bola dentro de dicho laberinto.

El siguiente apartado del documento muestra el estado del arte. En él se explica el trabajo previo al desarrollo de la herramienta, donde se comprobó el contexto en el que se crearía. En esta fase se buscaba respuesta a las preguntas: ¿existen ya herramientas con el mismo propósito?, si las hay, ¿qué características tienen?, ¿cuáles son sus limitaciones?... entre otras.

Los dos siguientes apartados explican cómo se ha desarrollado el trabajo en cuanto a la utilización de una determinada metodología y las diferentes etapas que se han llevado a cabo.

Después del apartado de etapas, se ha dedicado un apartado al uso de diferentes tecnologías durante el desarrollo.

También se dedica un apartado a las dificultades afrontadas a lo largo de todo el desarrollo.

Los siguientes apartados están más enfocados a la última fase del trabajo: muestran los resultados y las pruebas realizadas, así como el trabajo futuro.

Para finalizar el documento, se expone la conclusión del trabajo y seguidamente la bibliografía.

Estado del arte

Antes de comenzar con el desarrollo de la herramienta, se dedica un tiempo a investigar sobre la existencia de herramientas como esta.

Se ha comprobado que existen numerosas herramientas relacionadas con la que se quiere a desarrollar. En general, todas esas herramientas se pueden dividir en:

- Algoritmos que generan laberintos
- Juegos
- Modelos de impresión
- Generador de laberintos aleatorios

Algoritmos que generan laberintos

Más que una herramienta se puede ver como un posible componente de la herramienta. Existen varios algoritmos relacionados con la generación de laberintos, así como algoritmos que una vez generado el laberinto, calculen la solución del mismo.

Las características de la mayor parte de ellos son:

- Son algoritmos clásicos con fundamentos teóricos y han sido probados y utilizados numerosas veces: esto es una garantía, puesto que se sabe cómo funcionan y qué limitaciones tienen.
- Pueden tener diversas variaciones para adaptarse mejor a los requerimientos.

Un ejemplo es el algoritmo de Prim, que utiliza la técnica de generación por procedimientos, utilizada habitualmente en juegos y simulaciones, para crear laberintos difíciles.

Juegos

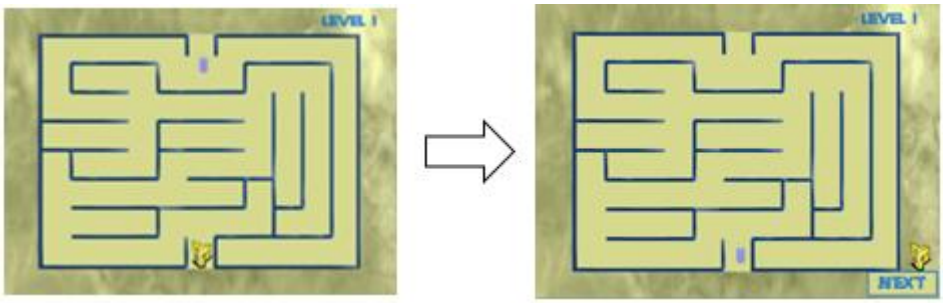
La cantidad de juegos que utilizan laberintos es abrumadora, algunas de las características comunes a todos ellos son:

- DIMENSIONES: Utilizan laberintos en dos dimensiones.
- COMPROBACIÓN: Se comprueba si se ha conseguido llegar al final del laberinto.
- SOLUCIÓN: Algunos disponen de un botón “solución” que muestra cómo salir del laberinto.
- DIFICULTAD: Suelen tener varios niveles según los cuales la dificultad del laberinto varía.
- PLATAFORMA: Abarcan muchas plataformas, como aplicaciones web o aplicaciones para Android.

Otra característica destacable, aunque poco común es la posibilidad de generar objetos con laberintos aleatorios que puedan ser impresos en una impresora 3D.

Un ejemplo de juego es el que se encuentra en el sitio: www.paisdelosjuegos.es/juego/laberinto en concreto “maze.html”. Es un juego bastante sencillo que solo contiene algunas de las características anteriormente mencionadas: tiene dos dimensiones y contiene el elemento de

dificultad y de niveles. En él se muestra el laberinto con dos objetos, un ratón y un queso, y consiste en llevar el ratón hasta el queso. Utiliza las flechas del teclado para mover el ratón y una vez que esto sucede, aparece un botón para poder pasar al siguiente nivel, en el que la dificultad aumenta.



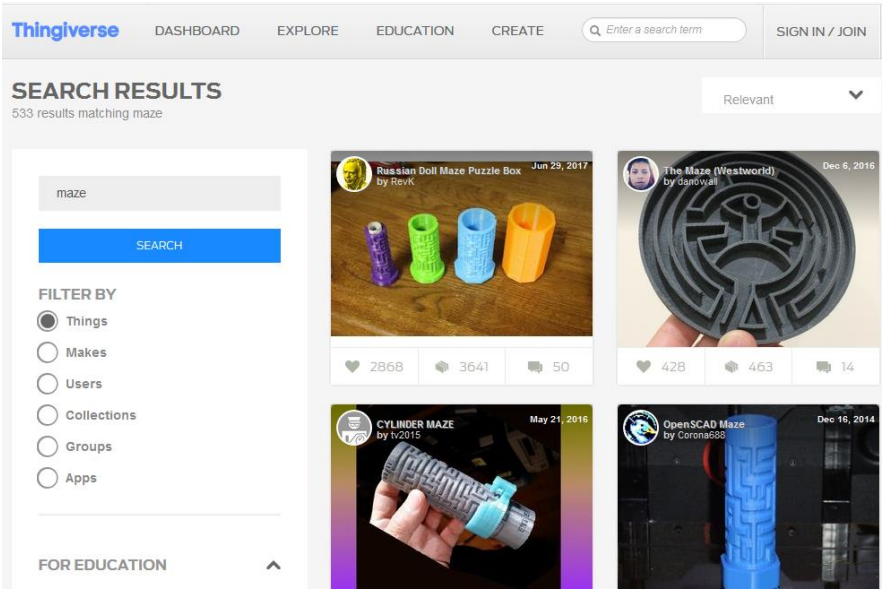
Buscadores de modelos de impresión

Se comprueba la existencia de herramientas que proveen al usuario con múltiples archivos de modelos tridimensionales que se pueden descargar e imprimir.

Las características de estas herramientas son:

- COSTE ECONÓMICO: Los buscadores son gratuitos y los archivos la mayoría también, aunque también los hay de pago.
- VARIEDAD: El usuario puede elegir el laberinto que más le guste de entre los que hay.
- CONTROL: El usuario no puede cambiar las dimensiones de un modelo dado ni buscar en función del tamaño del laberinto del modelo.
- PLATAFORMA: Se trata de páginas web.
- COMPARTIR: Además de descargar el usuario puede añadir sus propios modelos.

Uno de los ejemplos de este tipo de herramienta es la web https://www.thingiverse.com. En el aparece una barra de búsqueda en la parte superior de la pantalla donde se puede introducir lo que se busca, en este caso, laberintos (“maze” en inglés) y aparecen los resultados en un grid de imágenes de los resultados de la búsqueda. También permite utilizar diversos filtros para ajustar mejor la búsqueda.



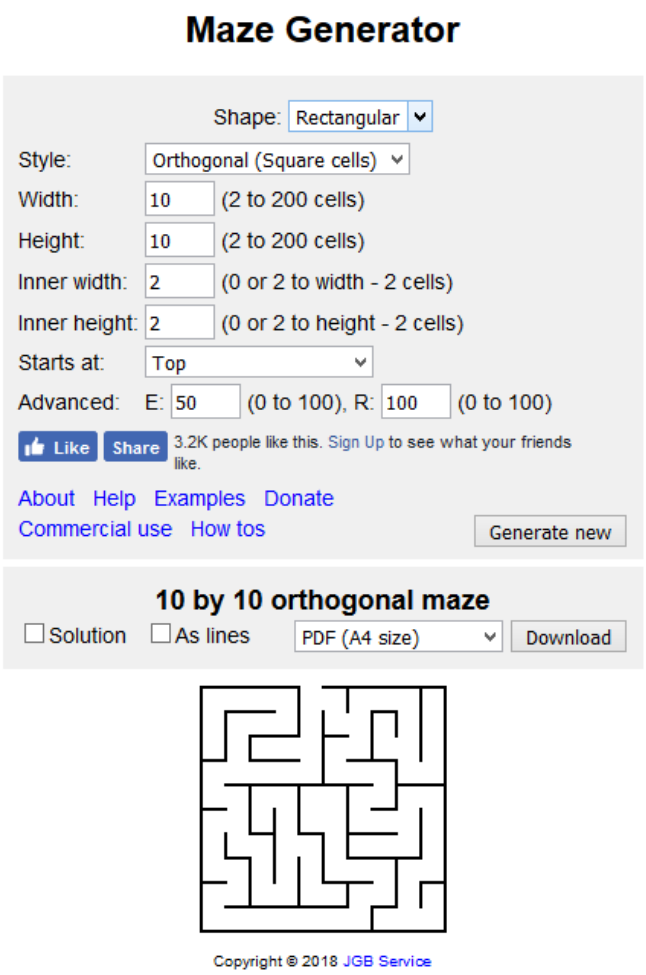
Generador de laberintos

Se ha comprobado que existen múltiples herramientas que generan laberintos. La mayoría de ellas generan laberintos en dos dimensiones, aunque también hay algunas que añaden una tercera dimensión.

Para los generadores de laberintos en dos dimensiones las principales características son:

- CONTROL: Permiten elegir al usuario las dimensiones del laberinto y controlar cuándo crear el laberinto, añadiendo un botón que genera el laberinto para ello.
- IMPRIMIBLES: Pueden ser imprimibles, en formato PDF, por ejemplo.
- SOLUCIÓN: Algunos pueden mostrar la solución
- ORGANIZACIÓN: Muestran dos zonas diferenciadas en la pantalla. Una sirve para mostrar los controles y la otra para mostrar el laberinto resultante.
- PLATAFORMA: La mayoría son aplicaciones web.
- VELOCIDAD: La mayoría son rápidas.
- COSTE ECONÓMICO: La mayoría son gratuitas.

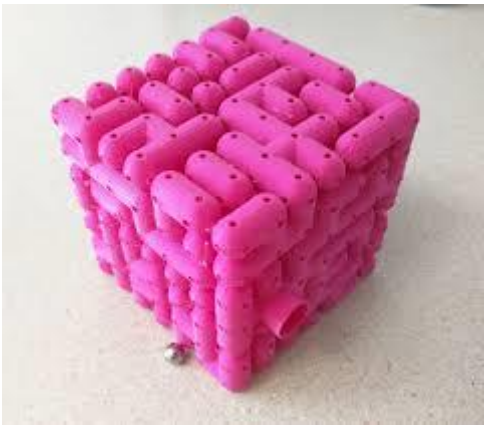
Un ejemplo bastante completo es el que se encuentra en http://www.mazegenerator.net/. En este ejemplo, el usuario puede elegir cómo será el laberinto y clicar en el botón de generar laberinto para mostrar por pantalla el laberinto generado aleatoriamente a partir de las características dadas por el usuario.



Por otro lado, para los generadores de laberintos en tres dimensiones las características son:

- COSTE ECONÓMICO: La mayoría son de pago.
- IMPRESIÓN 3D: Muy pocas permiten la impresión en 3 dimensiones.
- PLATAFORMA: No se han encontrado ejemplos de aplicaciones web.
- VELOCIDAD: La mayoría son rápidas.
- ALEATORIOS: La mayoría no son aleatorios. Son simples modelos para imprimir en 3D.

Como ejemplo se puede tomar el que se explica en esta página: https://conorpp.com/randomly-generating-3d-mazes-to-3d-print. En general, se trata de un programa que genera el archivo imprimible para un laberinto aleatorio de unas determinadas dimensiones.



La imagen anterior muestra el resultado de imprimir el archivo creado por el generador.

Recapitulando, en general, hay herramientas que generan laberintos pero no todas ofrecen todas las acciones posibles a realizar sobre ellos: la que muestra el laberinto, no puede imprimirlo; la que lo imprime, no lo muestra… La herramienta que se quiere crear combinará algunas de estas características: permitirá al usuario seleccionar las características del laberinto, crear laberintos de tres dimensiones con un laberinto aleatorio creado en función de la dificultad deseada por el usuario, visualizar el laberinto en la pantalla, poder mover una bola dentro del laberinto, generar el archivo imprimible…

En el siguiente apartado se explica la metodología con la que se ha puesto en marcha el trabajo.

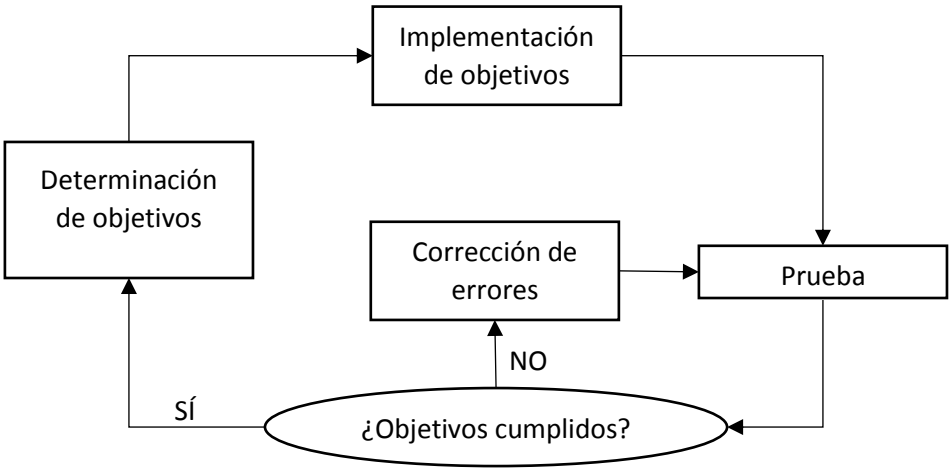
Metodología

El desarrollo ha estado marcado por el uso de una metodología ágil, en concreto, se podría decir que se ha utilizado Scrum.

Se ha utilizado Scrum porque es una de las metodologías ágiles más utilizadas y porque permite llevar a cabo un desarrollo iterativo. Esta característica hace que esta metodología se adapte muy bien a lo que se quiere conseguir con este trabajo puesto que no se quiere que sea algo estático.

Esto se debe a que la ausencia de programas que hagan exactamente lo que se quiere que haga esta herramienta y el desconocimiento inicial sobre la misma, provocan que se quiera tener la capacidad de ir cambiando los elementos según se van logrando resultados parciales durante el desarrollo.

Dada esta metodología, el trabajo se ha llevado a cabo en distintas fases como se explica en el siguiente apartado llamado "etapas del proyecto". El siguiente diagrama muestra cómo se han realizado estas etapas:



En cada una de esas etapas había uno o varios objetivos que cumplir. Se ha decidido que cada uno de los objetivos tuviera la misma prioridad a cada vez. Esto se debe a que el número de objetivos se ha mantenido siempre pequeño y los objetivos no tenían una gran complejidad en relación con el tiempo que se dedicaba a cada etapa. Es decir, los recursos disponibles para cada etapa aseguraban el cumplimiento de todos los objetivos. Además, a cada vez los objetivos a implementar eran independientes entre sí, por lo que la prioridad de unos objetivos sobre otros no tenía demasiado sentido durante el desarrollo.

El cumplimiento de un objetivo está definido como su funcionamiento sin errores durante las pequeñas pruebas realizadas durante el desarrollo.

Estas pruebas se explican en el apartado "pruebas" de este documento, concretamente en el subapartado "por el desarrollador".

Siguiendo la metodología Scrum, al final de cada etapa, se debe obtener una demostración gráfica del funcionamiento de los objetivos sobre la cual se hayan realizado las pruebas y en ella se detectarán las posibles mejoras.

Como marca la metodología, se realizaban reuniones al final de cada etapa para comprobar el logro de los diversos objetivos. Posteriormente, en función de los resultados obtenidos se marcaban los objetivos a cumplir para la siguiente etapa.

Al no tratarse de un trabajo en equipo, sino que se reducía a una sola persona, había únicamente dos roles en la reunión: el desarrollador y el cliente. El desarrollador era el estudiante y el cliente era el tutor del trabajo.

La forma en la que se ha realizado el trabajo ha estado determinada por la participación de la alumna en un programa de movilidad internacional. Este hecho ha provocado que haya sido esencial el uso del correo electrónico como herramienta de comunicación más que reuniones presenciales. Además, las fechas de las reuniones que han tenido lugar han tenido que hacerse en función de la disponibilidad, que ha estado marcada por el calendario de la universidad de destino (periodo de vacaciones escolares). Una de las consecuencias de esto es que las etapas en las que se dividió el trabajo no estuvieran bien balanceadas en cuanto a carga de trabajo a realizar en cada una de ellas.

Con todo ello, el calendario de reuniones presenciales, que muestra el tema principal de cada una de ellas en relación con cada fecha se puede recoger en la siguiente tabla:

Fecha	Tema Principal	Objetivos para la siguiente etapa
09/11/2017	Especificación y planificación	- Dominar las tecnologías necesarias para implementar la herramienta. - Creación de un diagrama de clases.
23/11/2017	Diagrama de clases	- Permitir al usuario introducir el largo y ancho del cubo. - Generar un cubo a partir de los valores introducidos por el usuario. - Generar una bola. - Permitir al usuario mover dicha bola. - Dado un laberinto, controlar el movimiento de la bola: se moverá solo cuando sea posible. - Advertir al usuario si un movimiento no se ha podido llevar a cabo.
20/02/2018	Laberinto 2D	- Añadir un botón para generar un cubo n*m. - Añadir un botón para generar laberinto. - Añadir un botón para poner la bola. - Añadir las paredes. - Modificar la definición de camino. - Implementación en tres dimensiones.
18/04/2018	Laberinto 3D y generación del archivo imprimible	- Permitir hacer zoom. - Al chocar que permanezca de otro color solo durante 3 segundo y después vuelva al color original. - Poder regenerar el cubo o el laberinto.

		- Mostrar el camino en rojo. - Modificar los controles: left-right-fordw.-backw.-up-down. - Generar un camino aleatorio. - Genera STL imprimible.
18/05/2018	Final	

Etapas del proyecto

Como ya se ha explicado, la realización del proyecto se ha llevado a cabo en etapas diferenciadas con uno o varios objetivos específicos en cada una de ellas. Estas etapas pueden contener subetapas cuando sus objetivos son más complejos o más numerosos.

La herramienta a crear se ha desarrollado de forma dinámica, de manera que se han llevado a cabo numerosas modificaciones entre una subetapa y la siguiente. Esto ha permitido mejorar la herramienta tras los sucesivos resultados parciales, detectando qué se podría mejorar y proponiendo una solución en la siguiente subetapa. Por ejemplo, el diseño de la interfaz ha sufrido numerosos cambios hasta la versión final.

Los siguientes subapartados explican cada una de las etapas ordenadas cronológicamente. En cada subapartado se profundiza más en detalles de cómo se ha desarrollado la herramienta.

1.- Especificación y planificación

Especificación

Se parte de que el objetivo de este proyecto es la creación de una herramienta para generar laberintos en 3D y poder imprimirlos en una impresora apropiada. Antes de nada, queda decidir cómo va a ser la herramienta, qué utilidades va a ofrecer y cómo se va a mostrar.

En concreto, vamos a crear una herramienta software, a modo de aplicación web, que permita al usuario interaccionar con ella para determinar cómo quiere que sea el laberinto resultante.

El software que se quiere desarrollar muestra al usuario una pantalla dividida en dos regiones. La primera de ellas, se muestra como un recuadro en la parte izquierda y sirve para que el usuario decida qué quiere hacer. En ella se encuentran todos los controles y todos los elementos con los que el usuario puede interactuar.

La segunda región está dedicada a mostrar el resultado de las acciones que realiza el usuario. Es decir, se podrá ver el cubo que genera la herramienta.

Se desea que el usuario tenga a su disposición los siguientes controles:

- Elección del largo, ancho y alto del cubo donde se creará el laberinto.
- Un botón “generar cubo” que crea el cubo a partir de los valores que se han seleccionado para cada una de las dimensiones.
- Un botón “generar laberinto” que crea un laberinto aleatoriamente en dicho cubo.
- Un botón “poner bola” que coloca una bola en la entrada del laberinto.
- Controles de movimiento en las 3 dimensiones para mover la bola creada.
- Un botón “generar STL” que crea el archivo imprimible para dicho laberinto.
- Poder hacer zoom y girar en la región donde se muestra el cubo.

Un flujo normal de utilización de la herramienta sería el siguiente:

1. El usuario introduce los tres valores que determinarán el largo, ancho y alto del cubo.
2. El usuario selecciona el botón “generar cubo”.
3. La herramienta crea el cubo y lo muestra.
4. El usuario selecciona el botón “generar laberinto”.
5. La herramienta crea el laberinto y muestra el cubo con el laberinto.

6. El usuario puede:
    a. Seleccionar el botón “poner bola”
    b. Seleccionar el botón “generar STL”

A partir de ahí, en el primer caso se podría continuar de la siguiente manera:

1. La herramienta crea la bola y la muestra en el primer hueco libre del laberinto.
2. El usuario utiliza los controles de movimiento.
3. Por cada control de movimiento la bola se mueve hacia la siguiente casilla si esta está libre o hace saber al usuario que no se ha podido mover.

En el segundo caso, la herramienta genera el archivo STL que se descarga en el equipo del usuario.

En ciertos casos, la herramienta tendrá que notificar de errores si la acción seleccionada no se ha podido llevar a cabo. Los casos en los que esto sucede son:

- Se pulsa el botón “generar cubo” cuando no se han introducido valores para sus dimensiones o los valores introducidos son erróneos (menores que 1).
- Se pulsa el botón “generar laberinto” sin haber generado el cubo primero.
- Se pulsa el botón “poner bola” sin que se haya definido un laberinto con anterioridad.
- Se pulsan los botones de movimiento de la bola sin haber generado la bola.
- Se pulsa el botón “generar STL” sin haber creado el laberinto.

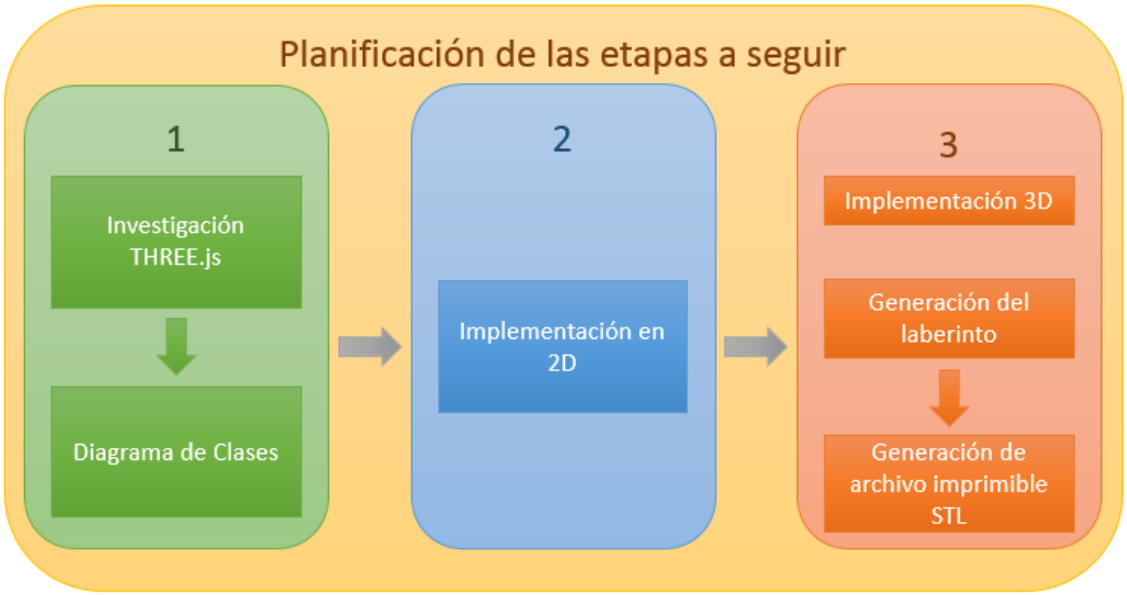
Se decide dejar abierto el diseño de la herramienta, permitiendo una evolución de algunos de los elementos a lo largo del desarrollo de la misma.

De esta manera, las decisiones de qué elementos utilizar para satisfacer las especificaciones del generador de laberintos se han tomado a lo largo del desarrollo.

En el siguiente apartado se explica cómo se ha organizado dicho desarrollo y se profundiza en cómo se han cumplido las especificaciones.

Planificación del trabajo

En el comienzo del trabajo, se decidió realizar pequeños objetivos de manera que profundizando sobre ellos se alcanzará el trabajo final. De esta forma, en esta parte se eligió el resto de las etapas en la que se dividiría el trabajo.



La figura anterior muestra tres subconjuntos claros en los que se dividirá el trabajo. Después de cada una de ellas se realizará una reunión. Se observa que dentro de cada una de ellas puede haber diferentes fases, tras las cuales no habrá reunión, pero sí un proceso de detección y corrección de errores.

Estas fases intermedias harán más fácil saber de donde proceden los errores en caso de haberlos y tener una mejor idea de en qué punto del desarrollo de la herramienta nos encontramos.

También se decidió qué tecnologías se utilizarían:

- La herramienta será una aplicación web (de manera que se han utilizado como lenguajes de programación html, css y javascript).
- Para crear y mostrar el cubo tridimensional al usuario se utiliza la librería THREE.js.

En el apartado "tecnologías utilizadas" se explica más a fondo cada una de ellas.

2.- Tecnologías y creación del diagrama de clases

Investigación de la librería THREE.js

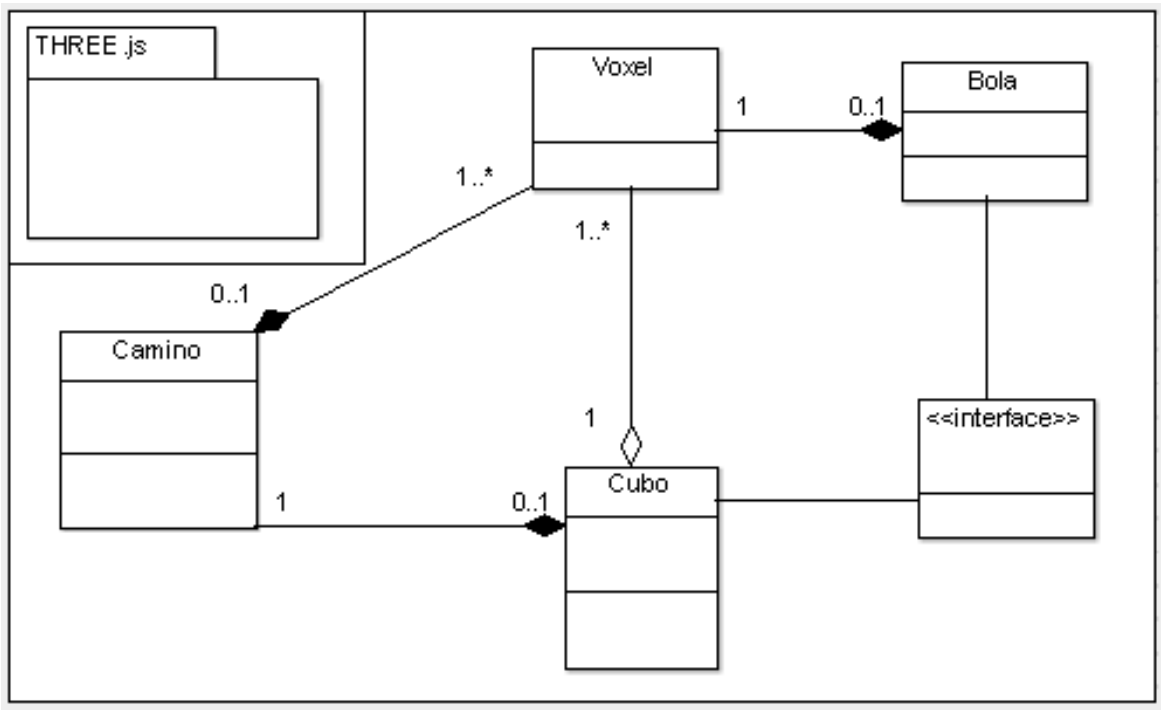
Los primeros días se han dedicado a investigar sobre la librería three.js, utilizando documentación disponible en Internet, un ejemplo de código y realizando pruebas para entender mejor el funcionamiento.

En el apartado de "Librería THREE.js", dentro de "Tecnologías utilizadas", se explica más a fondo en qué consiste esta librería y como se ha integrado en la herramienta.

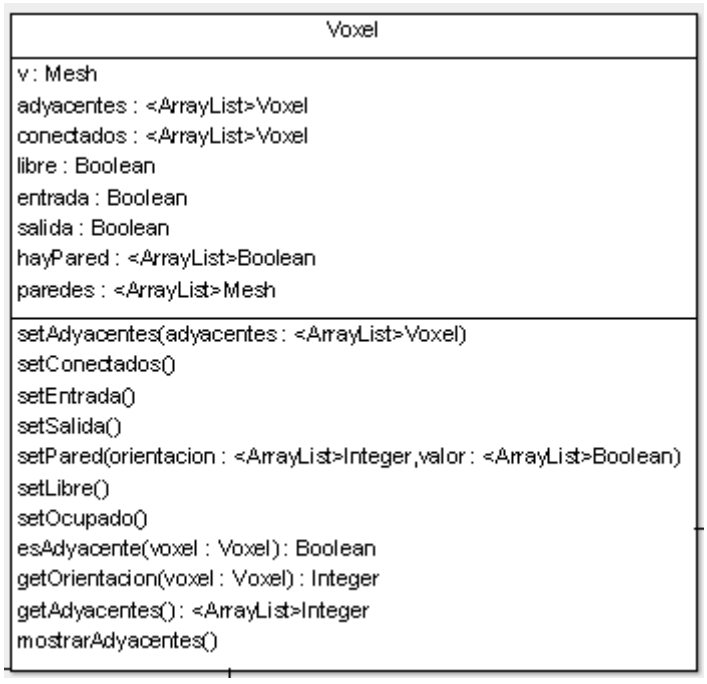
Diagrama de clases

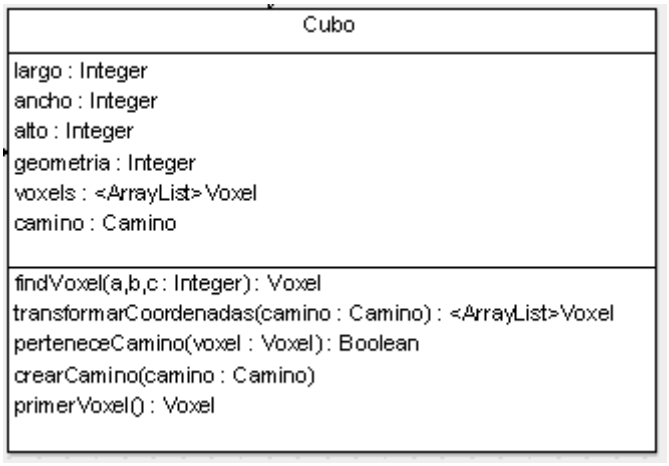
Para poder definir adecuadamente el programa y facilitar el desarrollo del mismo, se creó un diagrama de clases inicial. Este diagrama ha ido evolucionando posteriormente respondiendo a posibles mejoras que se han encontrado a lo largo de la elaboración del código. Aún así, las clases, las relaciones entre ellas y las funciones principales se han mantenido.

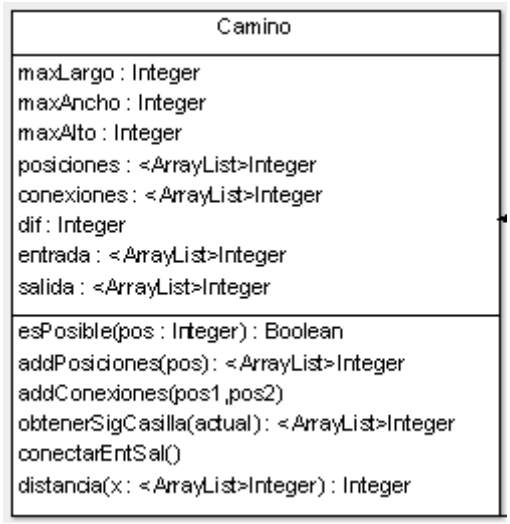
El diagrama que se ha creado se muestra en la siguiente figura:

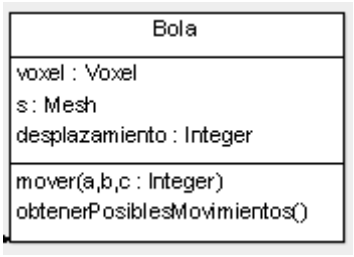


Donde cada clase tiene la siguiente estructura:









Y estas clases tienen un significado determinado en la aplicación:

- La clase voxel hace referencia a cada una de las posibles casillas.
- La clase cubo será la que contenga el conjunto de casillas y el laberinto.
- La clase camino hace referencia al laberinto.
- La clase bola es la bola que se moverá dentro del laberinto.

3.- Laberinto en 2D

Una vez se tenía una idea clara de qué se quería hacer a partir del diagrama de clases y con el conocimiento suficiente de la librería THREE.js, se llevó a cabo una primera implementación del programa que lejos de ser definitiva, es una primera aproximación sencilla que permitirá más adelante y tras sucesivos cambios obtener el objetivo global del proyecto.

En concreto, el objetivo de esta etapa es obtener como resultado un programa en el que el usuario elija las dimensiones de un rectángulo y se muestre en pantalla dicho objeto con un laberinto. Además, se debe permitir al usuario, dada una bola en el laberinto, que sea capaz de moverla. Obviamente, la bola solo se puede mover por el camino sin atravesar paredes. En caso de que el usuario intente un movimiento que no esté permitido, se le debe hacer saber.

En resumen, en esta parte se quiere conseguir básicamente dos partes de la implementación de la herramienta:

- La generación de una figura a partir de las dimensiones elegidas por el usuario:
    - Tener dos campos de entrada de datos para que el usuario introduzca el largo y el ancho de la figura que contendrá el laberinto.
    - Leer y validar esos datos en base a lo que necesitamos.
    - Mostrar por pantalla la figura

- El control del movimiento de la bola:
    - Mostrar en la pantalla, en concreto, en una de las casillas del laberinto, una bola.
    - Permitir que el usuario pueda mover la bola.
    - Comprobar que dicho movimiento es posible, es decir, si la casilla a la que se movería existe y es una de las casillas del camino.

Dados estos objetivos, de momento no se generará un laberinto aleatorio. Sin embargo, en su lugar, se pasará un vector de posiciones determinado al cubo y a partir de éste, se generará en el cubo un laberinto. Esto quiere decir que las casillas pertenecientes al camino, es decir, a dicho vector, estarán libres y el resto estarán ocupadas. De esta forma, la bola se solo se podrá mover a aquellas casillas que estén libres.

Además de ser necesario para controlar el movimiento de la bola, el hecho de utilizar el vector de posiciones en esta primera implementación y comprobar que funciona, tiene como motivación facilitar la etapa de generación de un laberinto aleatorio. Esto se debe a que su creación será completamente independiente en el sentido de que simplemente habrá que generar un vector aleatorio y pasárselo al cubo como el vector de posiciones utilizado anteriormente.

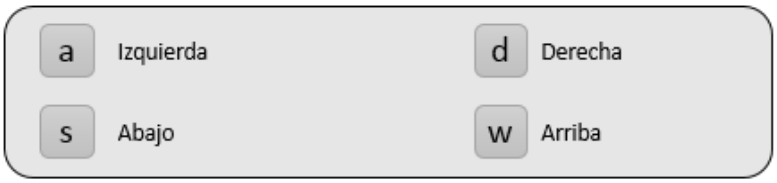
En esta etapa se diferencian dos aproximaciones ya que, al finalizar una primera implementación, se encontró la necesidad de realizar bastantes modificaciones que se recogen en una segunda implementación. Ambas aproximaciones se explican a continuación.

Primera aproximación

En esta primera aproximación nos centramos en implementar la creación del rectángulo a partir de los valores introducidos, la visualización del laberinto y el movimiento de la bola.

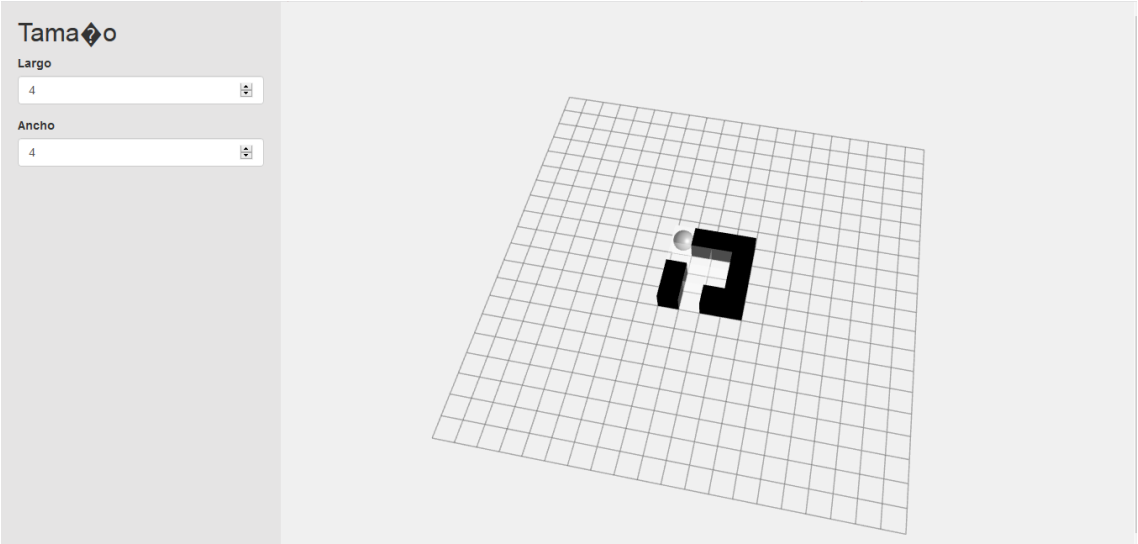
En la región de controles el usuario puede especificar el largo y el ancho. Cada vez que dichos valores cambian, se genera un rectángulo con las casillas del camino en blanco y el resto en negro. La región donde se muestra el laberinto se puede girar utilizando el ratón, pero por el momento no permite hacer zoom.

La bola se pone cuando se genera el cubo con el laberinto. De momento, se coloca en una de las casillas especificadas en el código en lugar de calcularla. El usuario puede utilizar las siguientes teclas para mover la bola:



Cuando el usuario intenta mover la bola hacia una casilla que no pertenece al camino, está no se mueve y se muestra un mensaje de error, a modo de alerta, al usuario.

El resultado de esta primera aproximación se muestra a continuación:



Tras obtener este resultado, se comprueba que se debe mejorar el movimiento de la bola de manera que se notifique al usuario de alguna manera más cómoda cuando el movimiento de la bola no es posible.

Por otra parte, y pensando en la siguiente etapa en la que habrá que generar el laberinto en tres dimensiones, puede resultar confuso utilizar teclas para moverse en cada una de las dimensiones.

Basándonos en las especificaciones, comprobamos que, además, la herramienta debería tener también un botón con el cual el usuario decide generar el cubo, otro para generar el laberinto y otro para colocar la bola.

Por estas razones, es necesario realizar otra aproximación para implementar el modelo en dos dimensiones. Todo lo mencionado anteriormente que falta por ser implementado, pasa a ser el conjunto de objetivos a lograr en la siguiente aproximación.

Segunda aproximación

Los objetivos que se tienen en este punto consisten en realizar modificaciones sobre el código existente para mejorar el resultado. En este sentido, mejorar el resultado significa que se obtendrá un resultado con más similitudes con la herramienta que se desea obtener al final.

Además de estos objetivos, se decide añadir el concepto de paredes al laberinto en lugar de diferenciar simplemente casillas. La razón por la que se toma esta decisión es que, basándonos en el diagrama de clases, al final la herramienta tendrá paredes. Como ya se ha conseguido controlar el movimiento de la bola de una manera simple (especificando si las casillas están libres u ocupadas), solo queda adaptarlo para que su movimiento dependa de la existencia de paredes entre la casilla en la que están y la casilla a la que se deberían mover.

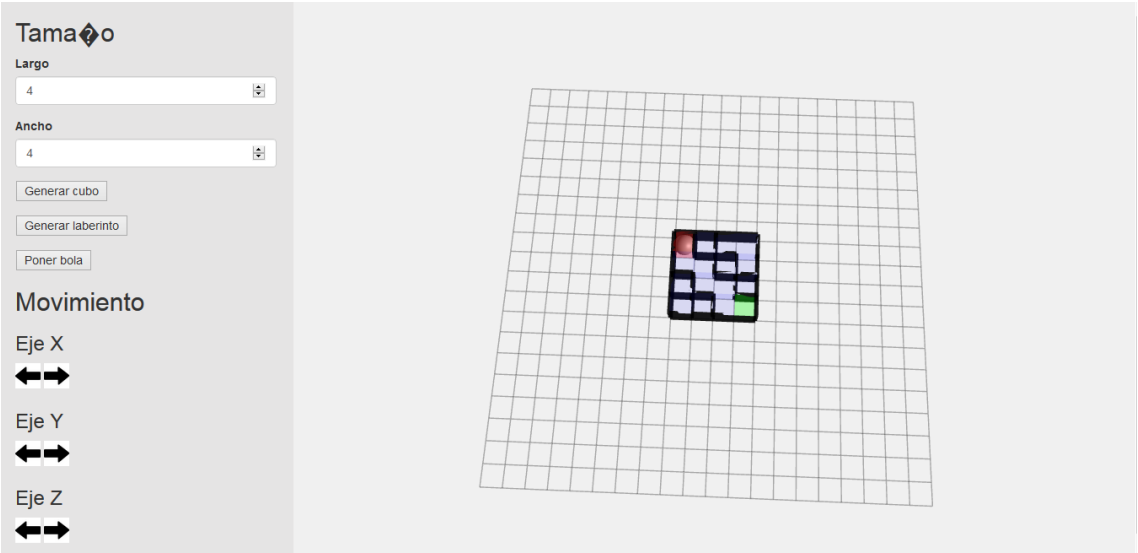
En la región de controles el usuario puede especificar el largo y el ancho de la misma forma que antes, pero para generar el cubo, el usuario debe pulsar en el botón de generar cubo que se muestra en esta misma región. Así mismo, se muestran los botones de generar laberinto y poner bola que el usuario deberá pulsar para poder generar el laberinto y la bola respectivamente.

Entonces, si el usuario pulsa el botón de generar el cubo, como ahora nos basamos en las paredes, se muestra el cubo con todas las casillas con paredes. Esta situación se prolonga hasta que se seleccione el botón generar laberinto. Cuando este cambio se produce, para todas las casillas que pertenecen al camino se comprueba si hay que quitar alguna pared, y en ese caso se quita. Para entender mejor el laberinto, se muestran la casilla de entrada y la de salida en un color distinto.

En esta aproximación, una vez que se ha generado un cubo o un laberinto, no se puede crear otro, aunque se utilicen los botones dedicados a este fin.

Cuando el usuario selecciona el botón de poner bola, ésta se coloca en la primera casilla del camino en lugar de una casilla especificada en el código. Para el movimiento, en lugar de utilizar el teclado, se utilizan unas flechas en el control de regiones que especifican el eje en el que tendrá lugar el movimiento (“X”, “Y” o “Z”) y el sentido. Se ha añadido el botón del tercer eje porque será necesario en la posterioridad, pero de momento no se utiliza. Cuando la bola no puede realizar un movimiento dado, esta cambia de color y no vuelve a su color propio hasta que realice un movimiento posible.

El resultado obtenido es el siguiente:



Se observa que sería mejor que se pudiera generar otro cubo si se desea, así como otro laberinto, por lo que en las implementaciones posteriores esto será algo a tener en cuenta.

Si nos ponemos en el lugar del usuario, comprobamos que, aunque los controles dedicados al movimiento de la bola han mejorado un poco respecto a la aproximación anterior, siguen siendo un poco confusos puesto que no se muestra en el cubo cuál es cada eje. Una mejora considerable sería añadir cada uno de los ejes diferenciándolos, por ejemplo, por colores.

Por otra parte, la bola cambia de color cuando no se puede mover, pero se queda con ese color hasta que se realiza un movimiento posible. Es preferible que la bola cambie de color únicamente durante un tiempo y después vuelva a su color.

4.- Laberinto en 3D y generación del archivo STL

Después de que el resultado de la etapa anterior funcione como se desea, se añade la tercera dimensión para conseguir un laberinto en 3D.

Como paso previo, se modifica la definición de camino, como se acordó en la reunión, de manera que en lugar de ser un vector de posiciones el camino pasa a ser un vector de posiciones junto con un vector de conexiones.

El vector de posiciones utilizado hasta la fecha, determina las casillas del camino por las que la bola puede pasar. En él se supone que habrá una conexión entre dos casillas si son sucesivas en el vector de posiciones.

Para facilitarlo, ahora el vector de posiciones solo ofrece información sobre las casillas del camino por las que se puede pasar. Para conocer qué casillas están conectadas, se utiliza el vector de conexiones. Cada elemento de este vector es un par en el que cada uno de los elementos son números que hacen referencia a la posición de la casilla en el vector de posiciones.

Los objetivos concretos de esta etapa son:

- La adaptación de la herramienta para añadir una tercera dimensión:

  - Añadir un campo de entrada de datos para que el usuario introduzca la altura del laberinto.
  - Leer y validar ese nuevo dato en base a lo que necesitamos.
  - Mostrar por pantalla la figura resultante.
  - Añadir movilidad a la bola en la tercera dimensión.

- La generación del laberinto aleatoriamente:
  - Creación e implementación del algoritmo a utilizar.
  - Adaptación para utilizar el laberinto generado en lugar del vector utilizado hasta ahora.

- Generar el archivo imprimible en formato STL:
  - Adaptar la visualización del laberinto.
  - Guardar el archivo en el ordenador del usuario.

Primera aproximación: Laberinto en 3D

Tras comprobar que la modificación de la definición de camino no afecta al funcionamiento del programa, se puede añadir la altura al laberinto.

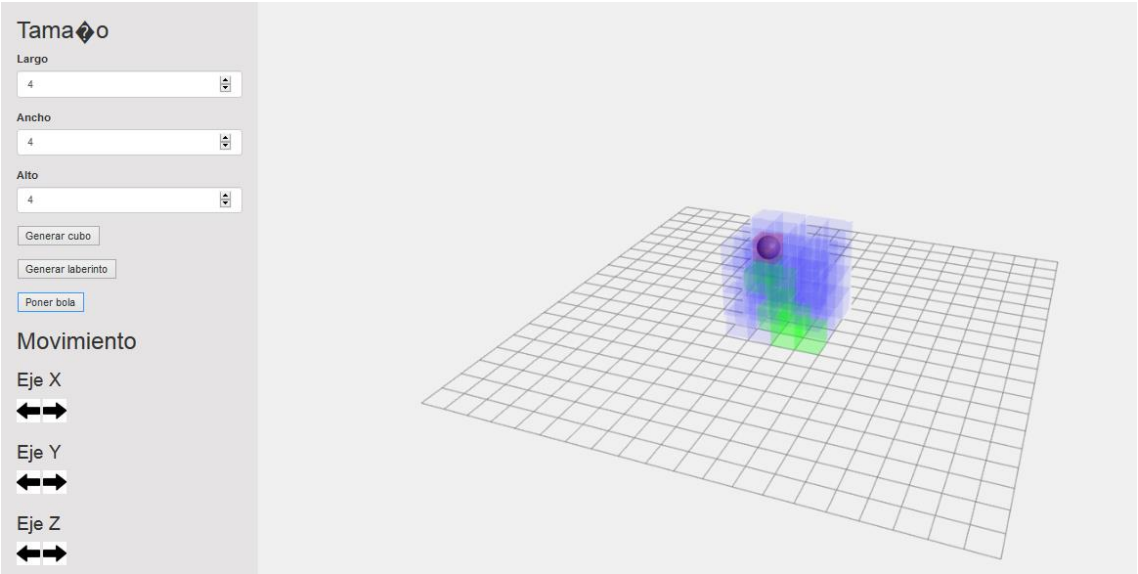
De momento no se tendrán en cuenta las necesidades encontradas en el resultado de la etapa anterior. El objetivo de esta fase será simplemente añadir una tercera dimensión y adaptar el funcionamiento del programa a esta nueva modificación.

Habrá que añadir una entrada de datos para que el usuario especifique la altura del cubo y mostrar por pantalla el cubo correspondiente.

En este punto nos damos cuenta de que, si utilizamos paredes para mostrar el cubo, al haber 3 dimensiones, va a ser muy incómodo para el usuario visualizar el laberinto. Esto se debe a que las paredes ocultarían lo que hay por detrás de ellas. Por ello, de momento, se decide cambiar la forma en la que se mostrará el laberinto: las casillas que no pertenezcan al camino se mostrarán de un color diferente a las que pertenecen al camino. Es decir, se utiliza la forma de mostrar el laberinto implementada durante la primera aproximación de la etapa anterior.

El resto de elementos se dejan tal y como se habían implementado en la segunda aproximación de la etapa anterior.

La apariencia de la herramienta en este punto es la que se muestra en la siguiente figura:



Segunda aproximación: Laberinto aleatorio

Cuando el laberinto en 3D funciona, se pasa a actualizar el programa de manera que el laberinto se genere automáticamente. Este es el objetivo principal de esta fase, pero también se recuperan los objetivos que quedan de la segunda aproximación de la etapa anterior: permitir generar otro cubo, laberinto o bola, mejorar los controles del movimiento de la bola y la notificación cuando un movimiento no se puede realizar.

Además, se quiere implementar otro de los requisitos de la aplicación del que no se ha hablado hasta ahora: poder hacer zoom en la región del cubo.

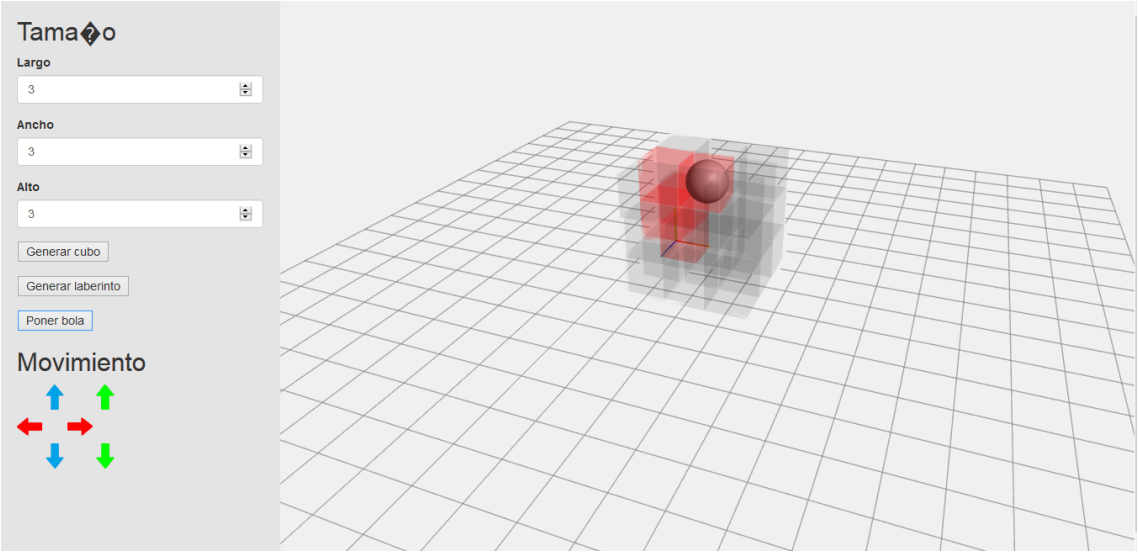
En este caso, cuando el usuario pulsa el botón de generar cubo, se borra el que estaba y aparece el nuevo. Lo mismo sucederá con el laberinto. Por último, también se permitirá generar otra bola, que siempre empezará en la primera casilla libre de camino.

Para mejorar los controles de movimiento de la bola, se añade un auxiliar para los ejes en una de las esquinas del cubo. Este auxiliar viene dado por la librería THREE.js y está formado por tres líneas, cada una en la dirección de cada eje a representar. Cada eje tiene asignado un color: rojo para el eje “X”, verde para el eje “Y” y azul para el eje “Z”. En la región de controles, se siguen utilizando flechas para mover la bola, con la diferencia de que en esta ocasión cada flecha tiene el color del eje con el que se asocia el movimiento de la bola. Así mismo, la disposición de las mismas cambia para que se parezca más a la apariencia de otros programas.

Para notificar al usuario de que un movimiento que ha intentado no se ha podido realizar, la bola cambiará de color como antes. Sin embargo, ahora, transcurridos cinco segundos la bola recuperará su color.

El usuario podrá alejar o acercar la región en la que se muestra el cubo utilizando la rueda del ratón.

La figura que se muestra a continuación permite saber cómo ha quedado la herramienta llegados a este punto:

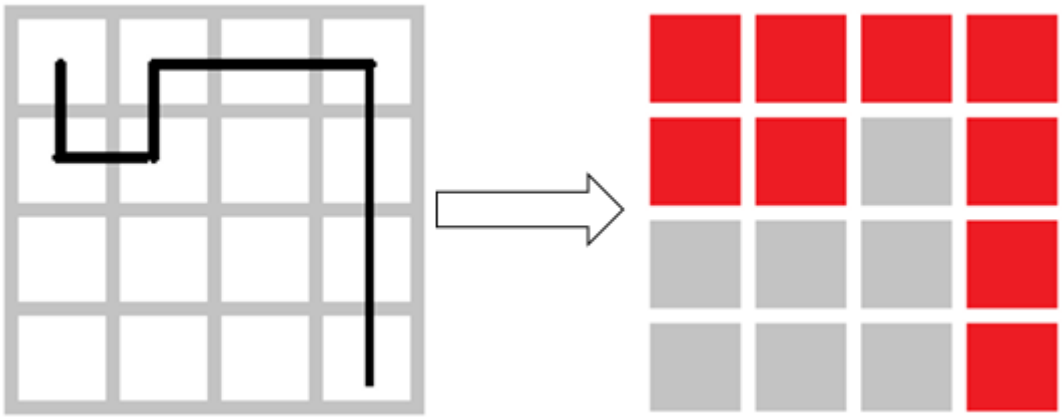


Se comprueba que se han cumplido los objetivos marcados para esta fase y se detecta que no interesa que se puedan poner varias bolas, por lo tanto, uno de los objetivos para la fase siguiente será el de impedir que se cree otra bola una vez exista una.

En cuanto a la apariencia, se detectan dos problemas. Por una parte, los ejes no se ven demasiado bien y sería mejor si se mostraran más grandes. Por otra parte, cabe destacar que tal y como se muestra el laberinto, puede haber una pared entre dos casillas y que el usuario no la vea y por lo tanto piense que no hay.

El segundo es un problema más grave en el sentido de que no queda claro por dónde va el camino. Recapitulando, en resultados parciales anteriores se intentó mostrar el laberinto como un conjunto de paredes en función del camino. Se comprobó que esto hacia prácticamente imposible distinguir por donde podía ir la bola ya que las paredes se solapaban.

En la última fase hasta el momento, se propone utilizar un color para la casilla si es una de las casillas por las que la bola puede pasar y otro color para aquellas por las que no puede. En cuyo caso surge el problema de que puede haber dos casillas por las que pueda pasar la bola, pero puede que haya una pared entre ellas.



La figura anterior muestra un ejemplo de cómo se mostraría un laberinto con la implementación utilizada hasta el momento. El laberinto de la izquierda es el que se quiere utilizar y el laberinto de la derecha es el que se mostraría al usuario. De manera que, según el laberinto de la izquierda, la bola no se puede mover de la casilla que está en la primera fila y la primera columna a la que está en la primera fila y la segunda columna porque debería haber una pared. Sin embargo, tal y como se muestra en el laberinto de la derecha, el usuario no puede saber que esa pared exista.

Por lo tanto, uno de los objetivos de la siguiente fase será encontrar una manera mejor de mostrar el cubo, de forma que se pueda distinguir si una bola puede o no pasar de una casilla a otra.

Para finalizar este apartado, cabe mencionar que también se ha detectado la necesidad de un elemento al que llamaremos dificultad. Esto se debe a la forma de implementar la generación del laberinto.

El algoritmo utilizado para la implementación es:

```
algoritmo generar_camino
    elegir_entrada
    elegir_salida
    conectar_entrada_salida
falgoritmo
```

Donde la función “elegir_entrada” devuelve como entrada del laberinto una posición aleatoria entre las del piso superior del cubo y la función “elegir_salida” devuelve como salida del laberinto una posición aleatoria entre las del piso inferior del cubo.

La tercera función, denominada “conectar_entrada_salida”, genera un camino aleatorio que conecta la entrada y la salida. Esta función requiere ser profundizada. Su algoritmo es:

```
algoritmo conectar_entrada_salida
    var actual = siguiente_casilla(entrada)
    var aux
    var cnx1 = entrada
    var cnx2 = añadir_posicion(actual)
    añadir_conexion(cnx1, cnx2)
    cnx1 = cnx2
    mientras (actual != salida)
        actual = siguiente_casilla(actual)
        cnx2 = añadir_posicion(actual)
        añadir_conexion(cnx1, cnx2)
        cnx1 = cnx2;
    fmientras
falgoritmo
```

Donde la función “siguiente_casilla” devuelve una casilla aleatoria entre las adyacentes a la casilla que se le pasa como parámetro de entrada, la función “añadir_posicion” añade (si no está) dicha casilla al vector de posiciones y devuelve la posición del vector donde se ha añadido;

y la función "añadir_conexion" añade la casilla y la casilla anterior al vector de conexiones cuando esa conexión no existe.

Entonces se comprueba que existe un problema considerablemente importante: puede suceder que, como se elige aleatoriamente la siguiente salida, haya casos en los que no se llega a la salida en un número de iteraciones pequeño. El tiempo de generación del laberinto, puede ser, por lo tanto, mayor al deseado.

Para solucionar este problema de una manera muy sencilla se propone la utilización de un número que determine el límite de número de iteraciones. Dicho número es el elemento que hemos mencionado anteriormente, llamado dificultad. Se utiliza para, llegado a ese número de iteraciones, calcular un camino directo hacia la salida. Con lo que quedaría el siguiente algoritmo para la función que conecta la entrada y la salida:

```
algoritmo conectar_entrada_salida
    var actual = siguiente_casilla(entrada)
    var aux
    var t = 0
    var cnx1 = entrada
    var cnx2 = añadir_posicion(actual)
    añadir_conexion(cnx1, cnx2)
    cnx1 = cnx2
    mientras ((actual != salida) y (t<dificultad))
        actual = siguiente_casilla(actual)
        cnx2 = añadir_posicion(actual)
        añadir_conexion(cnx1, cnx2)
        cnx1 = cnx2
        t++
    fmientras
    calcular_camino_directo_salida
falgoritmo
```

Donde la función "calcular_camino_directo_salida" obtiene el camino más corto desde la posición en la que esta en ese momento hasta la posición correspondiente a la salida. Va añadiendo las casillas del camino al vector de posiciones y las sucesivas conexiones al vector de conexiones.

Por lo tanto, en la siguiente fase habrá que implementar este algoritmo, junto con las funciones necesarias para que funcione, y añadir los controles necesarios para que el usuario pueda determinar dicho valor.

Tercera aproximación: Generación del archivo STL

Llegados a esta fase, debemos ser capaces de generar un archivo ".stl" a partir del contenido de la escena, que en este caso será el cubo con el laberinto.

La primera parte de esta fase se ha centrado en investigar sobre cómo generar este tipo de archivos a partir de la escena generada en nuestra herramienta. Se decide utilizar el componente STLExporter de la librería THREE.js que se explica más adelante en el apartado "tecnologías utilizadas", en concreto en el subapartado "STLExporter".

Este componente consta de una función que genera un archivo imprimible a partir de los objetos que se encuentre en la escena, que en la herramienta se muestra en la segunda región de la pantalla. Lo que se quiere imprimir son todas las paredes existentes en el laberinto, pero ya se decidió anteriormente que para el usuario esa vista era inadecuada. Por esta razón, se decide implementar la generación de otra vista en la que solo se muestren las paredes (no interesa imprimir las casillas porque éstas deben ser huecas para que la bola pueda pasar). El usuario podrá cambiar de una vista a otra: la vista utilizada hasta ahora será la vista "general" y la otra será la vista "esqueleto". El usuario solo podrá imprimir si se encuentra en la vista que muestra las paredes.

Además de este objetivo principal, también deberemos tener en cuenta las mejoras necesarias que se han descubierto en la fase anterior. La primera de ellas es impedir que se cree otra bola una vez exista una. La segunda mejora a realizar es mostrar los ejes de coordenadas más grandes. Por otro lado, hay que añadir el elemento dificultad para limitar el número de iteraciones para la generación del laberinto, de manera que sea especificado por el usuario. Por último, habrá que encontrar una manera en la que mostrar el cubo para que el usuario pueda ver por dónde pasa el camino del laberinto dejando a un lado la ambigüedad que detectada en el cierre de la fase anterior.

Las tres primeras mejoras se implementan satisfactoriamente en esta fase y no precisa ninguna explicación dado su carácter trivial.

Para llevar a cabo la cuarta mejora, se opta por mostrar el laberinto prácticamente como en la fase anterior, pero con una diferencia: esta vez se mostrarán alguna de las paredes. Se podría decir que el resultado es una mezcla de la implementación propuesta en la segunda aproximación de la primera implementación (llevada a cabo durante la segunda aproximación de la tercera etapa) y la tercera implementación (la correspondiente a la segunda aproximación de la cuarta etapa).

En concreto, se mostrarán las casillas del laberinto con un color diferente al resto de casillas, para que sean fácilmente diferenciables, y se mostrarán las paredes de las casillas en las que pueda haber ambigüedad. En cada casilla del camino, se comprobarán cada una de las casillas adyacentes a dicha casilla. En el caso de que una casilla adyacente pertenezca también al camino, habrá que tener en cuenta si existe una conexión entre ambas casillas. Entonces, se mostrará la pared correspondiente. En caso contrario no.

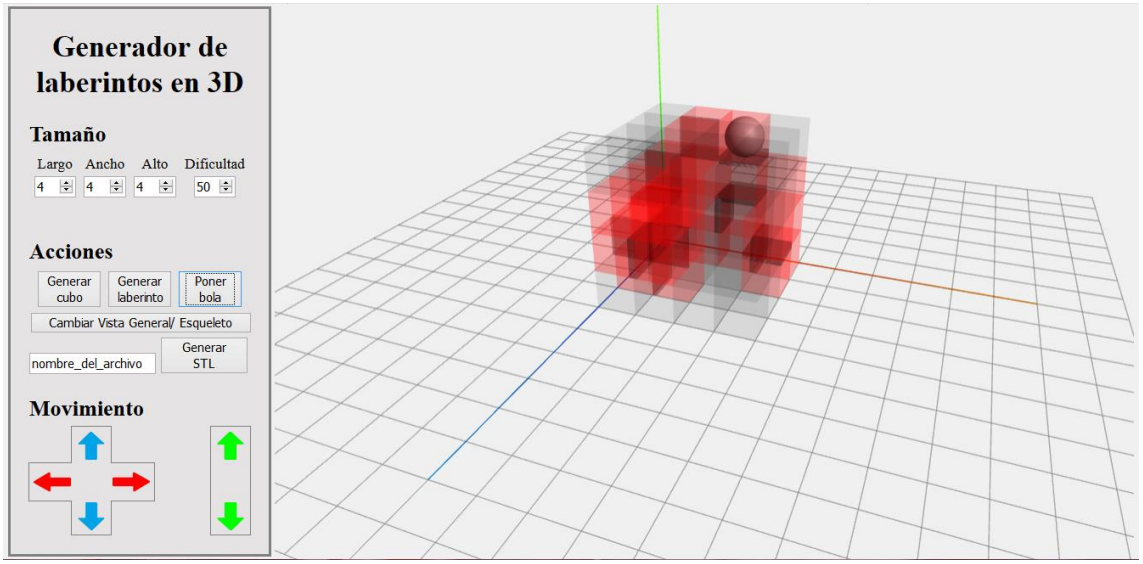
También se decide al inicio de la fase como se mostrarán al usuario todas las opciones. Llegados a esta fase, no quedan por implementar más acciones que el usuario pueda realizar. Por este motivo, se aprovecha esta fase para definir el diseño final de la interfaz de la herramienta. La región de controles contendrá el título de la herramienta y posteriormente una subregión para definir el tamaño del cubo, una segunda subregión para las acciones que se le permiten realizar al usuario y una última subregión para los controles de movimiento de la bola.

Se desea que los campos en los que el usuario elige el tamaño sean lo más pequeños posibles (que quepan un máximo de tres cifras) y se muestren todos en una misma fila.

Los botones de la segunda subregión también ocuparán menos y deberán estar divididos en dos partes: una para la parte de generación de objetos (cubo, laberinto o bola) y otra para las acciones requeridas por la acción de generar el archivo imprimible.

También se toma la decisión de que el usuario pueda elegir el nombre a utilizar para el archivo imprimible. Por lo tanto, en la parte dedicada a la generación de dicho archivo, se desea colocar un campo de texto y, a continuación, en la misma fila, el botón de generar STL. Como hemos visto inicialmente, se implementarán dos vistas que el usuario podrá cambiar, por lo tanto, antes del botón de generar STL, se mostrará un botón en la línea anterior para cambiar de vista: si está en la general cambiará a la de esqueleto y al revés.

El resultado de la realización de esta fase es el siguiente:



Esta versión de la herramienta cumple con todos los requisitos, la apariencia parece apropiada y parece funcionar. Para asegurar su buen funcionamiento se lleva a cabo la etapa de "detección y corrección de errores".

Además de estas etapas, tras las cuales siempre se ha llevado a cabo una reunión presencial, a lo largo del desarrollo y sobre todo tras completar la implementación de la última etapa, se realiza un proceso de detección y corrección de errores.

Detección y corrección de errores

Este proceso se lleva a cabo antes de la última reunión y su finalidad es asegurar que la herramienta no tenga errores.

Para comprobar que no hay errores en la herramienta, se realizan diversas pruebas que se pueden dividir en dos tipos: las pruebas realizadas por el desarrollador y las pruebas realizadas por usuarios potenciales.

Estas pruebas se explican en detalle en la sección de pruebas de este documento.

Tecnologías utilizadas

Este apartado se dedica a la explicación de las tecnologías que se han necesitado para la creación de la herramienta.

HTML, javascript, css

Al tratarse de una aplicación web, se utilizan como lenguajes de programación HTML, JavaScript y CSS en combinación para controlar qué contenido se muestra, cómo interactuar con el usuario y cómo se muestra el contenido.

No es necesario explicar más en detalle estas tecnologías puesto que no se utiliza nada diferente a cualquier otra aplicación web.

Librería THREE.js

Una parte esencial de la herramienta a desarrollar es la necesidad de mostrar objetos tridimensionales por pantalla. Para poder crear este tipo de programas, como se utilizan gráficos en tres dimensiones, utilizaremos el hardware específico para gráficos, es decir, se necesita acceder a la unidad de procesamiento gráfico (GPU).

Como nuestra herramienta tomará la forma de una aplicación web, se utiliza JavaScript, como se ha explicado anteriormente en el documento. Existe una API que permite acceder a la GPU utilizando JavaScript. Esta API se llama WebGL y con ella se puede renderizar un objeto 3D en una web (en un elemento <canvas>).

El inconveniente es que se trata de una programación a bajo nivel, muy compleja y la cual requiere de muchas líneas de código para cada acción. Por lo tanto, para facilitar este proceso, se utiliza la librería THREE.js cuyo objetivo es facilitar el uso de WebGL.

Esta librería es mucho más simple y fácil de utilizar y se encarga de generar internamente el código WebGL necesario. Se necesitan muchas menos líneas de código y es más fácil de aprender puesto que consta de numerosos métodos y componentes pre-generados.

El código de esta librería se encuentra en un repositorio en GitHub, al que se puede acceder a través del enlace https://github.com/mrdoob/three.js/, desde que su autor, Ricardo Cabello, lo creara en abril de 2010.

Algunas de las clases y funciones más importantes que se han utilizado en el código, merecen una mención especial.

Una de las clases esenciales ha sido la clase "Scene". Esta clase permite crear una escena para luego poder determinar qué se va a mostrar. En el código se utiliza para colocar en ella el laberinto que se va a generar, así como la bola.

```
scene = new THREE.Scene();          // Crea la escena donde se mostrarán los elementos
                                    // como el laberinto y la bola
```

Para poder especificar qué objetos se muestran en dicha escena, utilizamos la variable que la contiene y la función "add". Por ejemplo, para añadir la bola a la escena, se utilizaría el siguiente código:

```
scene.add(bola.s);
```

Donde bola es la clase que hemos creado para definir la bola que se moverá en el laberinto, y s es uno de sus atributos que hace referencia a la geometría esférica utilizada.

Para poder quitar los objetos que están en la escena, se utiliza la función "remove", que funciona igual que la función "add" pero, en vez de mostrar el objeto lo oculta.

```
scene.remove(bola.s);
```

Esta tabla muestra de una manera más resumida qué función concreta de la librería se utiliza, junto con ejemplos de cómo se aplica en concreto en el código de la herramienta (aplicación) y la explicación de lo que hace (propósito):

Función	Aplicación	Propósito
Scene()	scene = new THREE.Scene();	Crea la escena donde se mostrará el laberinto

scene.**add(**object**)**	scene.add(line);
	scene.add(plane);
	scene.add(ambientLight);
	scene.add(directionalLight);
	scene.add(cubo.voxels[i].v);
	scene.add(axis);
	scene.add(cubo.voxels[i].paredes[j]);
	scene.add(bola.s);
		Permite mostrar en la escena creada todos los objetos que se necesitan. Por una parte, las lucen la línea y el plano para determinar cómo será la escena y por otro lado los voxels del cubo, las paredes y la bola para el laberinto.

scene.**remove(**object**)**	scene.remove(cubo.voxels[i].v);
	scene.remove(cubo.voxels[i].paredes[j]);
	scene.remove(bola.s);
	scene.remove(cubo.voxels[i].paredes[j]);
	scene.remove(v);
		En función de qué queremos mostrar, se eliminan los elementos que no queremos que aparezcan.

Otra clase muy utilizada es la clase "PerspectiveCamera" que permite determinar cómo se ve la escena. En concreto, es uno de los modos de proyección más utilizado en renderizado en 3 D y, como se explica en la documentación disponible en la página web de la librería (que se explica en la dirección https://threejs.org/docs/#api/cameras/PerspectiveCamera), se trata de intentar imitar cómo ve el ser humano. Algunas de las funciones utilizadas se muestran en la siguiente tabla:

Función	Aplicación	Propósito
THREE.PerspectiveCamera(cdv, proporción, cerca, lejos**);**	camera = new THREE.PerspectiveCamera(45, canvasWidth() / canvasHeight(), 1, 10000);	Crear el objeto cámara para especificar posteriormente las propiedades de la misma para ajustar la forma en la que se verá la escena.
camera.**position.set(** x, y, z**)**	camera.position.set(500, 800, 1300);	Especifica dónde se va a colocar la cámara.
camera.**lookAt(**[x, y, z]**)**	camera.lookAt(new THREE.Vector3());	Especifica la dirección hacia la cuál mira la cámara.

Para crear un "grid" o cuadrícula sobre el cual colocar el laberinto se utiliza la clase "Geometry" (https://threejs.org/docs/#api/core/Geometry). Se decide utilizar una cuadrícula para mejorar la visualización al usuario. Por una parte, servirá para orientarse: saber si la escena esta girada hacia algún lado, conocer qué es arriba y qué es abajo… Por otra parte, dará información sobre dónde empieza una casilla y dónde acaba (porque el tamaño de los cuadrados de la cuadrícula es el mismo que el de las casillas del laberinto).

Función	Aplicación	Propósito
Geometry()	var geometry = new THREE.Geometry();	Crea un objeto geometría que puede ser definido una vez creado indicando sus vertices, caras…

geometry.vertices.push([x, y, z]**)**	for (var i = - size; i <= size; i += step) {
	geometry.vertices.push(new THREE.Vector3(-size, 0, i));
	geometry.vertices.push(new THREE.Vector3(size, 0, i));
	geometry.vertices.push(new THREE.Vector3(i, 0, -size));
	geometry.vertices.push(new THREE.Vector3(i, 0, size));
	}
		Para insertar vértices en la geometría.
		Como queremos que sea una cuadrícula, se recorre el total de la cuadrícula insertando los cuatro vértices correspondientes a cada celda de la cuadrícula.

Además de estas clases, se utilizan otras que permiten crear atributos de otras clases o mejorar la visualización de la escena.

Función	Aplicación	Propósito
Vector3(x, y, z**)**	new THREE.Vector3(-size, 0, i)	Se utiliza para crear un vector tridimensional para definir algunos atributos o propiedades de las clases mencionadas anteriormente.
LineBasicMaterial({lista}**)**	var material = new THREE.LineBasicMaterial({ color: 0x000000, opacity: 0.3, transparent: true });	Permite generar un material que lleva información sobre el color, la textura, el tipo de material que es… para poder para definir algunos atributos o propiedades de las clases mencionadas anteriormente.
LineSegments(geometry, material**)**	var line = new THREE.LineSegments(geometry, material);	Sirve para crear líneas que unirán diferentes vértices de la geometría. Se utiliza para mostrar las líneas de la cuadrícula creada anteriormente.
Raycaster()	raycaster = new THREE.Raycaster();	Objeto que se utiliza para permitir al usuario mover la escena pinchando con el ratón en ella y arrastrando.
PlaneBufferGeometry(ancho, alto, anchoSegmento, altoSegmento**)**	var geometry = new THREE.PlaneBufferGeometry(1000, 1000);	Permite crear una figura plana de manera eficiente. Se utiliza para crear el plano que se corresponde con la cuadrícula anteriormente mencionada.

geometry.**rotateX(**angulo**)**	geometry.rotateX(-Math.PI / 2);	Gira la geometría creada tanto como se le indique en el ángulo que se le pasa como parámetro. Se utiliza simplemente por motivos de estilo, para que la cuadrícula se muestre rotada.
Mesh()	var plane = new THREE.Mesh(geometry, new THREE.MeshBasicMaterial({ visible: false }));	Crea el objeto tridimensional a partir de la geometría y el material deseados. Se utiliza para obtener la cuadrícula en tres dimensiones que se pueda mostrar en la escena.

Una vez se han creado todos los objetos tridimensionales que se quieren mostrar por pantalla, y, para ello, se han añadido a la escena creada, hay que renderizar dicha escena. Para ello, como se explico al principio de este apartado, es necesario utilizar la unidad de procesamiento gráfico. Con este fin, se hace uso de la clase "WebGLRenderer" (que se explica una vez más en la página de documentación: https://threejs.org/docs/#api/renderers/WebGLRenderer), que se encargará de realizar todas las acciones pertinentes a partir de las propiedades que elijamos.

Función	Aplicación	Propósito
WebGLRenderer()	renderer = new THREE.WebGLRenderer({antialias: true});	Crear el objeto que renderizará la escena con el atributo "antialias" con valor verdadero para mejorar la visualización.
renderer.**setClearColor(**color**)**	renderer.setClearColor(0xf0f0f0);	Para especificar el color con el que se borrarán los objetos de la escena en caso de que se quieran ocultar.
renderer.**setPixelRatio(**ratio**)**	renderer.setPixelRatio(window.devicePixelRatio);	Permite determinar cuál va a ser la proporción entre los píxeles para obtener una mejor visualización.
renderer.**setSize(**ancho, alto**)**	renderer.setSize(canvasWidth(), canvasHeight());	Para adaptar lo que se quiere mostrar al elemento "canvas" donde se va a mostrar.

Para los controles, como por ejemplo, poder girar en la escena o hacer zoom, se utiliza la clase "OrbitControls" (se puede consultar la información sobre esta clase en la documentación de three.js en https://threejs.org/docs/#examples/controls/OrbitControls):

Función/Atributo	Aplicación	Propósito
OrbitControls()	controls = new THREE.OrbitControls(camera, renderer.domElement);	Crea el objeto que permitirá girar la cámara alrededor de la escena.
enableDamping	controls.enableDamping = true;	Para permitir que haya una amortiguación cuando se hace uso de los controles, por ejemplo, al hacer zoom. Se utiliza simplemente por motivos de estilo.

dampingFactor	controls.dampingFactor = 0.25;	Simplemente especifica cuánta amortiguación habrá al utilizar los controles.
enableZoom	controls.enableZoom = true;	Hace posible acercar o alejar la escena. Es necesario para uno de nuestros objetivos en el que el usuario debe de ser capaz de hacer zoom utilizando la rueda del ratón.

STLExporter

Esta tecnología que se incluye en la librería THREE.js merece una mención especial ya que con ella se ha logrado una de las características de nuestra herramienta: generar un archivo imprimible.

Es una de las herramientas que ofrece la librería THREE.js para poder exportar contenido. En este caso el contenido que se exporta es el conjunto de elementos que se hallen en la escena en un determinado momento. Estos objetos se exportan en un formato llamado STL que se utiliza en impresión en 3D, que es para lo que lo utilizaremos en este caso, pero que se puede usar también en prototipado y en fabricación asistida por ordenador.

Los archivos STL contienen únicamente la información sobre la superficie de la geometría a imprimir. Para ello, utiliza la subdivisión de dicha superficie en triángulos y la información sobre los vértices de dichos triángulos y sus normales. Esta información se representa haciendo uso del sistema de coordenadas cartesianas. En impresión en 3D, la información contenida en este archivo se utiliza para construir el objeto como un conjunto de trozos de dicho objeto.

En la herramienta creada, cuando el usuario selecciona el botón de generar STL, se produce una llamada a la función generarSTL del código creado, que básicamente, llama a la función saveSTL de STLExporter. Esta llamada devuelve el objeto STL que se almacena en una variable.

Como paso previo a la llamada, se comprueba que en la pantalla se este mostrando la versión imprimible del laberinto, que difiere de la versión con la que el usuario juega por motivos de visualización (como se explicó en el apartado de etapas). Si la vista no es la que debe ser, la función devolverá 0 y no se generará el archivo.

En caso de que la vista sea la que debe ser, antes de realizar la llamada saveSTL, se coge el nombre que el usuario haya querido dar al archivo desde el campo dedicado a este fin. Para finalizar, se utiliza la función saveAs de FileSaver, con la variable que contiene el objeto STL y el nombre dado. Este último paso permite al usuario guardar en su ordenador el archivo STL generado por la herramienta.

```
function generarSTL() {      // Generar archivo STL
    if (!vista) return 0;
    var nombre = $('input.nombre').val() + ".stl";
    var blob = saveSTL(scene);
    saveAs(blob, nombre);
    return 1;
}
```

Por lo tanto, llegados a este punto se observa la necesidad de utilizar la librería FileSaver para guardar el archivo generado.

FileSaver

Esta librería se halla en uno de los repositorios de GitHub, en concreto en el que se menciona en el apartado de bibliografía. La función de esta librería es permitir guardar archivos con un nombre dado. Algunos navegadores permiten esta función, pero otros muchos no. Por este motivo, para que permitir al usuario guardar el archivo independientemente del navegador que utilice, se decide utilizar esta librería.

Como se explica en el archivo “readme”, esta librería sirve para guardar archivos en el cliente y es perfecto en caso de necesitar generar archivos.

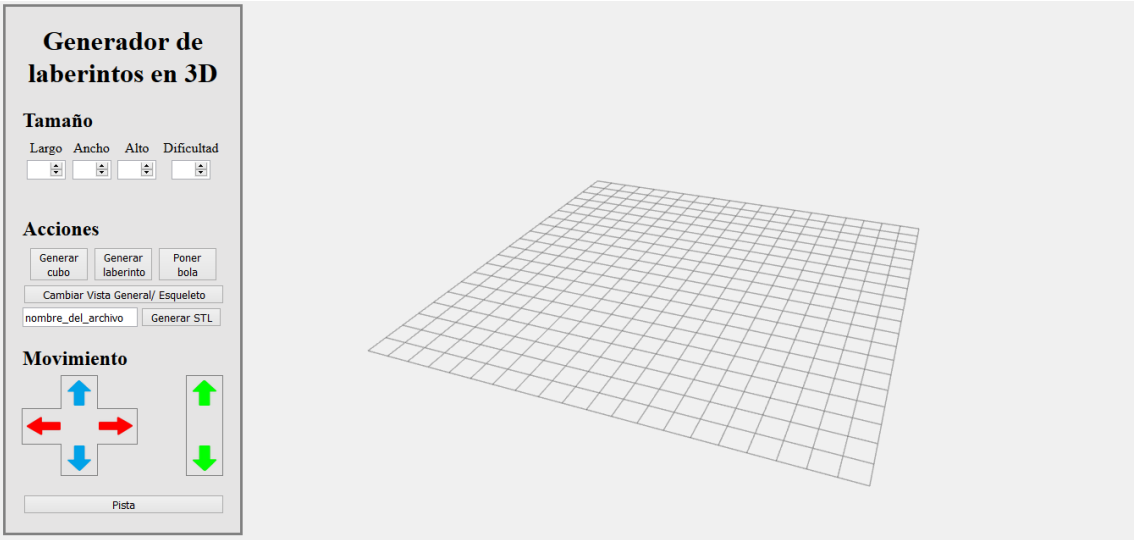
En el apartado anterior dedicado a STLExporter se explica cómo se ha utilizado concretamente en la herramienta desarrollada.

Resultados

En este apartado se muestra cómo ha quedado finalmente la herramienta. Para ello se utilizarán capturas de la herramienta en los diferentes estados en los que se puede encontrar.

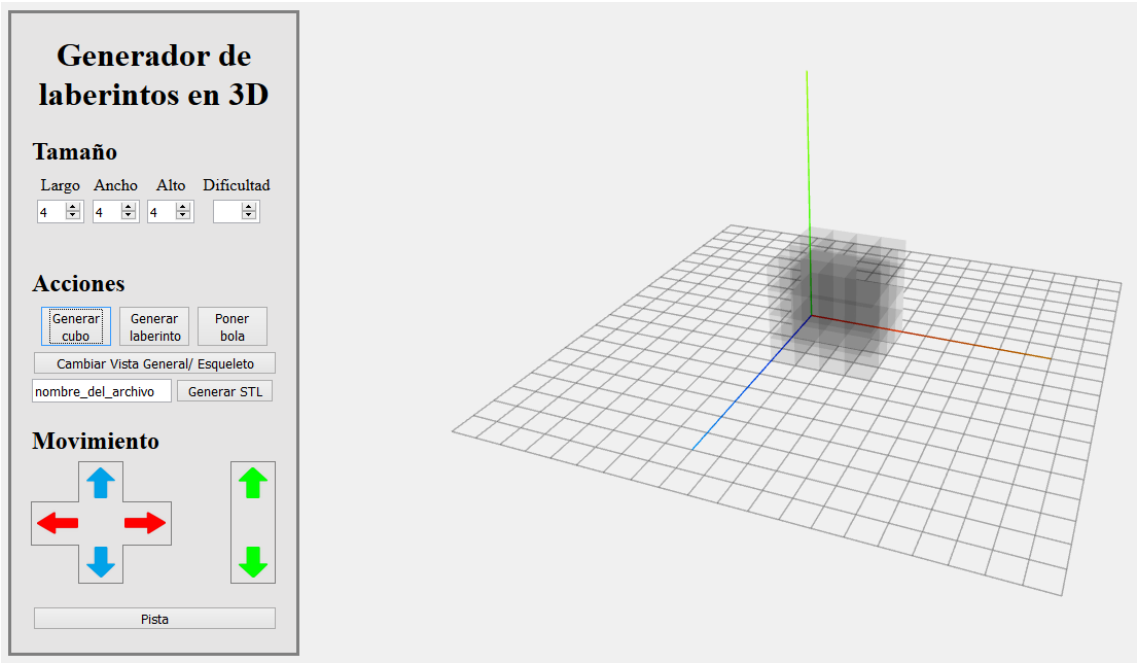
Como se trata de una herramienta que necesita la interacción del usuario, se propone un ejemplo de qué acciones podría realizar el usuario y cómo cambiaría la apariencia de la herramienta en función de esas acciones.

En un primer momento, la herramienta se muestra de la siguiente manera:

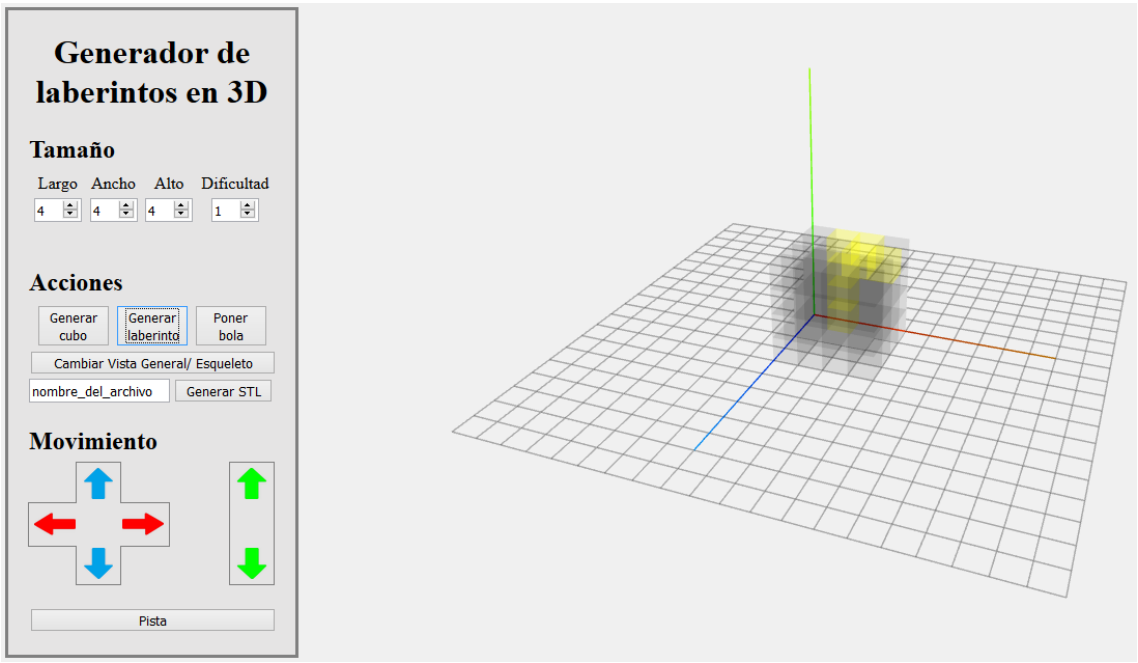


Donde se muestran las dos regiones diferenciadas que se quieren tener. A la derecha la región dedicada a mostrar el laberinto, cuando se haya creado. A la izquierda, la región dedicada a los controles, que se muestra mejor en la siguiente imagen:

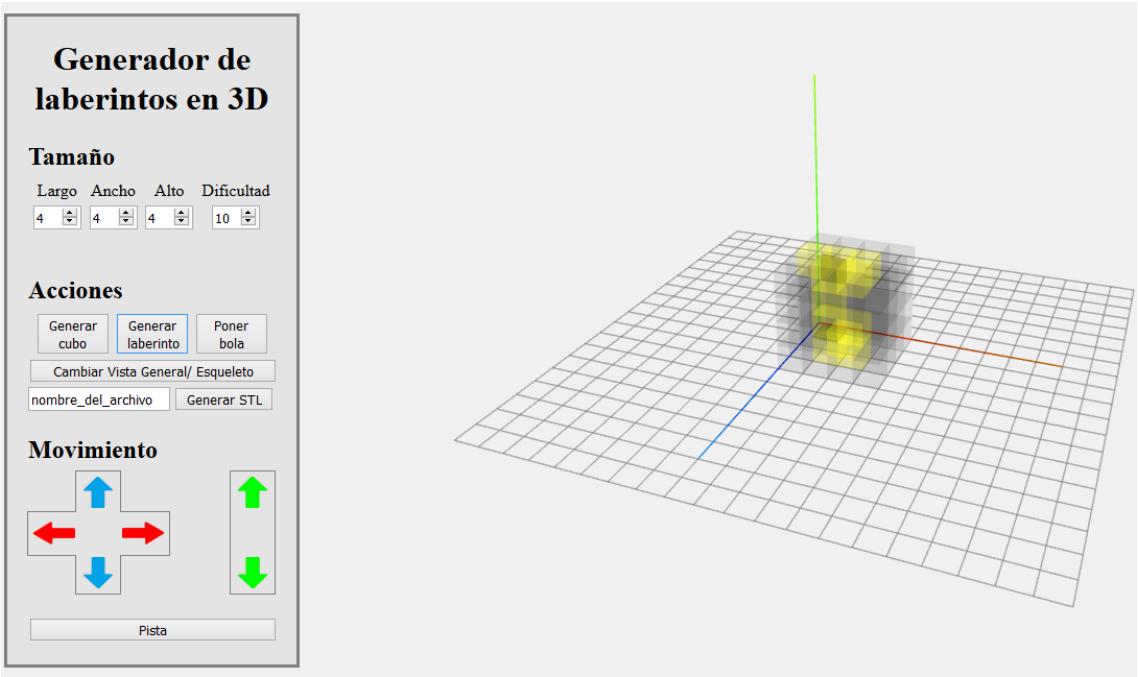
Una vez seleccionamos las longitudes para cada una de las dimensiones del cubo (por ejemplo, 4 x 4 x 4) y pulsamos el botón de “generar cubo”, se obtiene el siguiente resultado:



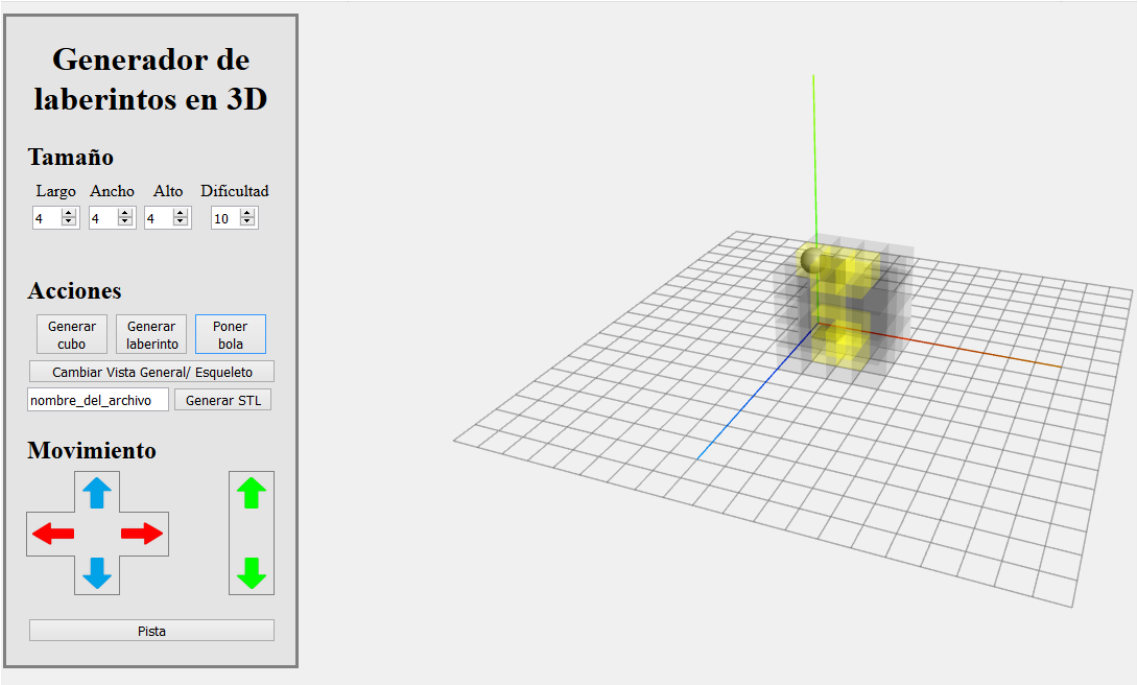
Después, elegimos una dificultad, por ejemplo 1 y pulsamos el botón “generar laberinto” y obtendríamos el siguiente laberinto:



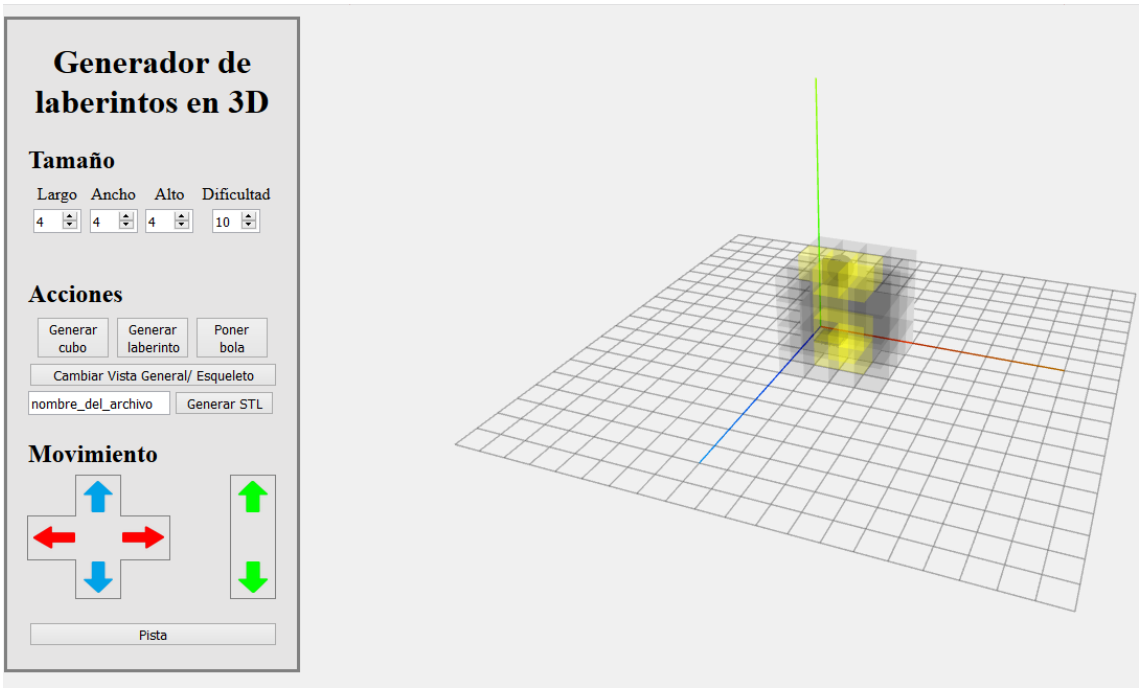
Si probamos con una dificultad mayor, por ejemplo 10, y volvemos a generar un laberinto, obtenemos un laberinto de mayor dificultad:



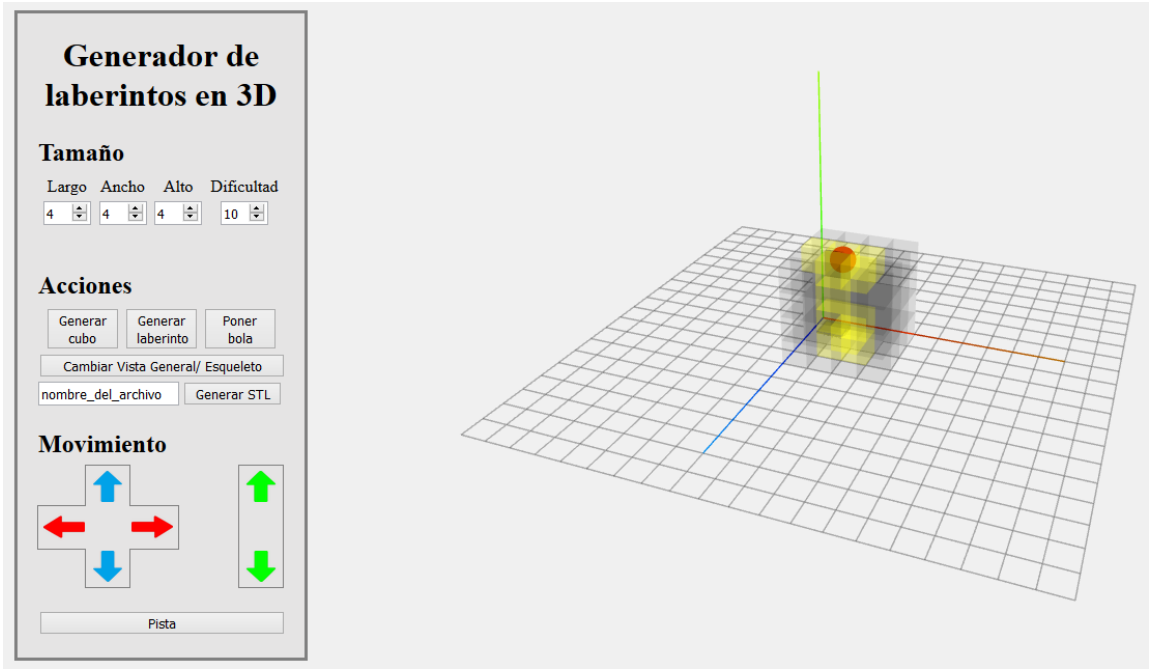
Cuando se pulsa el botón “poner bola” se muestra el mismo laberinto con una bola en una de sus casillas (la casilla de entrada). Esto se muestra en la siguiente imagen:



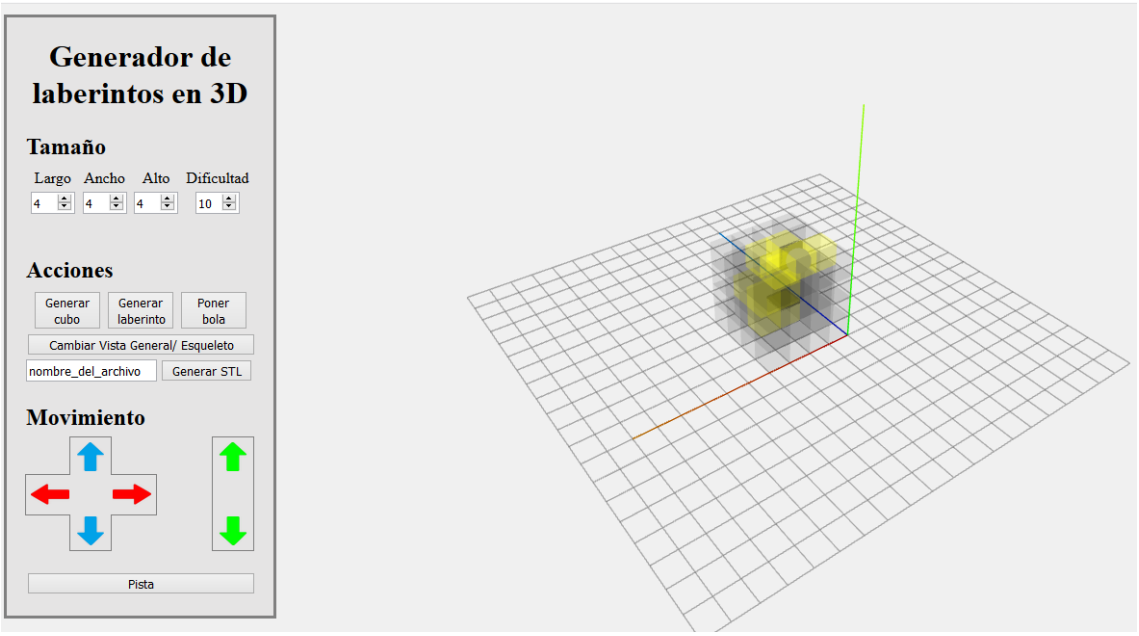
Si intentamos mover la bola a una casilla libre esta se mueve a dicha casilla:



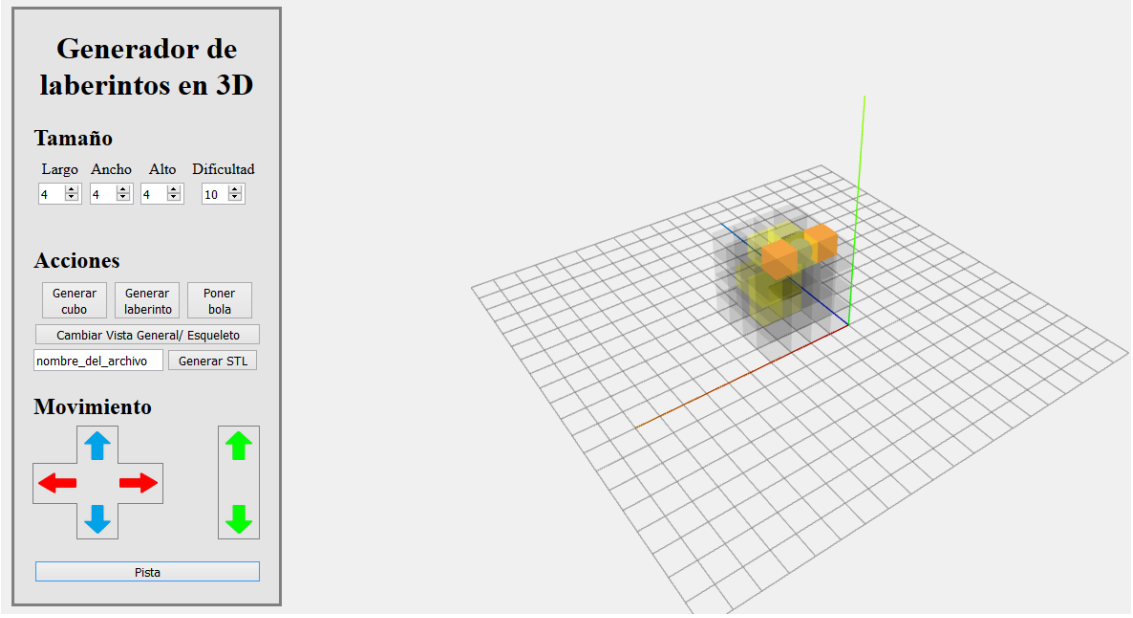
Si por el contrario, intentamos moverla a una casilla para la cual existe una pared en medio, no se moverá y notificará del error cambiando de color:



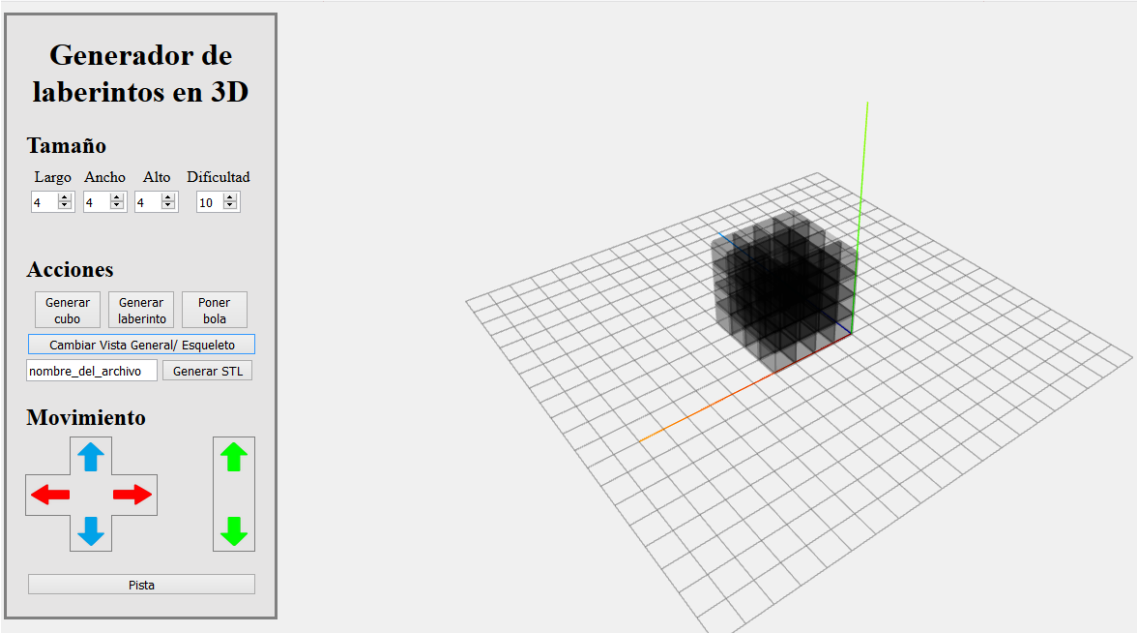
Utilizando el ratón, podemos girar la cámara para ver el laberinto desde otro ángulo. Para ello, pinchamos en la región de la derecha y arrastramos el ratón hasta estar en el ángulo deseado. Por ejemplo, se podría ver desde el ángulo que se muestra en esta imagen:



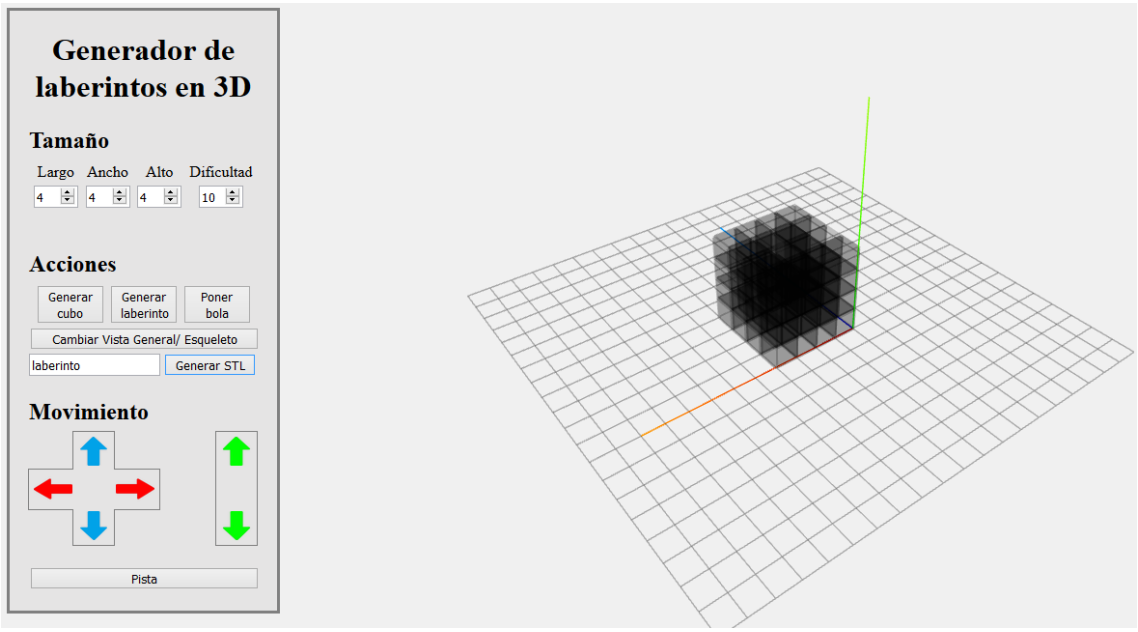
Si no somos capaces de distinguir bien hacia que casilla se puede mover la bola, podemos pulsar el botón "pista" para obtener los posibles movimientos:



Una vez tenemos creado el laberinto que queremos imprimir, podemos utilizar el botón "cambiar vista" para acceder a la vista "esqueleto" desde la cual se podrá generar el archivo STL. A continuación, se muestra el resultado de la vista "esqueleto" para el laberinto dado. Como se puede observar en la siguiente imagen, se generan todas las paredes que se deben imprimir:



Cuando se quiere generar el archivo imprimible, desde la vista “esqueleto” se puede elegir un nombre para el archivo, en este caso, “laberinto” y se pulsa el botón “generar STL” tras lo cual la descarga se inicia automáticamente.



Si el usuario intenta generar el archivo imprimible el archivo desde la vista general, la herramienta notificará de un error e informará al usuario de que debe cambiar la vista para poder generar el archivo STL.

Si se intenta crear una bola habiendo ya una creada o sin haber un laberinto creado, también se informa del error:

Otro posible error puede surgir si se intenta generar el laberinto sin haber un cubo o sin haber definido la dificultad:

Finalmente, también se informa del error que puede surgir al intentar generar un cubo sin haber definido previamente los valores para cada una de las dimensiones, como se muestra en la siguiente pantalla.

Pruebas

Como se ha mencionado anteriormente, se han llevado a cabo dos tipos de pruebas: las llevadas a cabo por el desarrollador y las realizadas por usuarios de prueba.

Por el desarrollador

El objetivo de las pruebas realizadas por el desarrollador es detectar errores o posibles mejoras sobre los resultados parciales obtenidos. Esto se realiza de esta manera para ir corrigiendo los errores en cada implementación y evitar arrastrarlos en las sucesivas implementaciones. De esta manera, no solo se facilitará saber la procedencia de los errores sino también corregirlos en un periodo de tiempo menor.

Estas pruebas eran muy simples y tenían lugar una vez que se creía haber terminado de implementar un determinado objetivo. Como la herramienta devuelve un resultado en concreto en función de la entrada del usuario, la simpleza de las pruebas se debe a que éstas se basan en introducir valores como lo haría un usuario y comprobar si la herramienta se comporta como se esperaba.

En el caso de las dimensiones del laberinto, por ejemplo, al tratarse de un conjunto infinito de valores, se probaba con dos o tres valores que representaran un subconjunto de todos los posibles valores de entrada.

Por ejemplo, suponemos que estamos intentando implementar la generación de un cubo de una casilla de alto y donde el largo y el ancho vienen especificados por el usuario. Para las dimensiones del cubo las entradas posibles son el conjunto de números reales. Se podrían diferenciar tres subconjuntos: en el primer subconjunto alguna de las entradas es un número menor que uno; en el segundo subconjunto todas las entradas son mayores o iguales que uno y todas las entradas son iguales; y, en el tercer subconjunto, todas las entradas son mayores o iguales que uno y las entradas no son todas iguales. Las posibles pruebas podrían ser:

N.º de prueba	Entradas	Valores	Subconjunto	Resultado Esperado
1	Largo	0	1	Notificación de error
	Ancho	2		
2	Largo	4	1	Notificación de error
	Ancho	-1		
3	Largo	3	2	Muestra un cubo 3x3x1
	Ancho	3		
4	Largo	6	3	Muestra un cubo 6x5x1
	Ancho	5		
5	Largo	2	3	Muestra un cubo 2x8x1
	Ancho	8		
6	Largo	9	2	Muestra un cubo 9x9x1
	Ancho	9		
7	Largo	-10	1	Notificación de error
	Ancho	-6		

Por otra parte, se probaban diferentes combinaciones en las que pulsar los diferentes botones existentes para comprobar si la herramienta respondía debidamente.

Por usuarios de prueba

Las pruebas llevadas a cabo por usuarios de prueba tuvieron lugar únicamente después de haber terminado la última etapa de desarrollo.

El objetivo de estas pruebas es comprobar si un usuario es capaz de interactuar con el programa y obtener feedback de su experiencia.

Como paso previo, se decide diseñar cómo van a ser las pruebas. Algunos de los aspectos que se quieren tener en cuenta son:

1. Conocer si se entiende bien para qué sirve la herramienta.
2. Conocer si una vez se sabe para qué sirve la herramienta, se puede saber cómo utilizarla.
3. Conocer si sabiendo cómo funciona, su manejo resulta agradable para el usuario.
4. Conocer cómo de fácil es para un usuario resolver un laberinto generado por la herramienta.
5. Conocer la opinión final del usuario tras su experiencia.

Para cada uno de esos aspectos, se emplean diversas estrategias:

1. Mostrar al usuario la primera pantalla y pedirle que responda a las siguientes preguntas:
    a. ¿Qué crees que hace la aplicación?
    b. ¿Cómo crees que se podría utilizar?

2. Explicar al usuario en qué consiste la herramienta, permitirle interaccionar con ella durante tres minutos y pedir que responda a las siguientes preguntas:
    a. ¿Qué controles ves en la pantalla?
    b. ¿Qué crees que hace cada uno de ellos?
    c. ¿Qué acciones permite realizar la herramienta?

3. Explicarle al usuario cómo funciona y pedirle que realice ciertas acciones para comprobar cuánto tarda en realizarlas.
    a. Genera un cubo de 6x8x7
    b. Genera un laberinto de 3x3x3 con dificultad 5
    c. Genera un laberinto de 4x4x4 con dificultad 2 y muévete a través del laberinto.
    d. Genera un laberinto de 4x4x4 con dificultad 100 y muévete a través del laberinto.

4. Comprobar cuánto tiempo tarda el usuario en salir del laberinto en los casos anteriores.

5. Pasarle un formulario final al usuario con las siguientes preguntas:
    a. ¿Qué te parece la aplicación?
    b. ¿Qué es lo que más te ha gustado la aplicación? ¿Por qué?
    c. ¿Qué es lo que menos te ha gustado de la aplicación? ¿Por qué?
    d. ¿Crees que falta algún control? ¿Cuál?
    e. ¿Crees que sobra algún control? ¿Cuál?
    f. ¿Cambiarías la visualización de algún elemento? ¿Cuál/cuáles? ¿Cómo?
    g. ¿Cómo mejorarías la aplicación?

Los usuarios objetivo de las pruebas, son personas jóvenes que hayan tenido experiencia utilizando aplicaciones web, aunque no tienen porque tener experiencia utilizando aplicaciones que tengan que ver con gráficos en 3D.

Usuario 1

El primer usuario es un joven de 25 años con estudios superiores que se dedica a la informática.

Los resultados para cada una de las preguntas mencionadas anteriormente han sido los siguientes:

1. a. La aplicación crea laberintos tridimensionales.
   b. Para hacer laberintos en 3D en los que jugar. Para crear laberintos para ratones de laboratorio.

2. a. Las flechas (arriba, abajo, derecha, izquierda, encima y debajo). El ratón para mover el mapa y para hacer zoom.
   b. Las flechas sirven para mover la bola dentro del laberinto y el ratón sirve para girar el laberinto y hacer zoom.

3. El usuario ha sido capaz de realizar todas las acciones, pero en la segunda acción, en lugar de pulsar primero el botón de "generar cubo" ha pulsado primero "generar laberinto". Después se ha dado cuenta del error y ha rectificado.

4. El usuario tarda muy poco en realizar todas las acciones a excepción de la última. El usuario ha tardado bastante en encontrar la solución y ha tenido que hacer uso de las pistas.

5. Tras pasarle el formulario final al usuario se obtienen las siguientes respuestas para cada una de las preguntas:

   ¿Qué te parece la aplicación?
   La aplicación es interesante, porque muestra cómo se crean internamente puzles en 3 dimensiones. Normalmente los puzles son en 2D, y son muy fáciles, pero este es complejo, ya que siempre hay que tener en cuenta como está posicionado el cubo con respecto a los ejes del mapa.

   ¿Qué es lo que más te ha gustado la aplicación? ¿Por qué?
   Lo que más me ha gustado es la posibilidad de generar el cubo en 3 dimensiones con una impresora 3D para mover mi propia bola en la realidad.

   ¿Qué es lo que menos te ha gustado de la aplicación? ¿Por qué?
   Las pistas no funcionan como pensaba y cuando el laberinto está en modo difícil, no ayudan gran cosa. A veces es difícil distinguir la posición de la bola, dado que como está entre cubitos, no se ve muy bien. Los controles son fáciles de entender y las opciones para programar el cubo son muy intuitivas.

   ¿Crees que falta algún control? ¿Cuál?
   Me gustaría poder cambiar el color de la bola, para elegir un color que me parezca más visible y personalizado.

¿Crees que sobra algún control? ¿Cuál?
No creo que sobre ningún control.

¿Cambiarías la visualización de algún elemento? ¿Cuál/cuáles? ¿Cómo?
El texto de "Pista" lo cambiaría por "Posibles movimientos", ya que no te dice hacia donde debes mover la bola para llegar al final, sino dónde puedes mover la bola en cada momento que pulsas el botón.

¿Cómo mejorarías la aplicación?
Creo que para mejorar la aplicación sería interesante un modo inmersivo, en el cual puedas moverte por el laberinto en primera persona, de tal forma que no se vea a través de los cubos y debas aprender la ruta para encontrar la salida.

Cabe destacar que durante la prueba el usuario ha hecho los siguientes comentarios:

<<Muy mal, tendrías que tener un limitador al elegir las dimensiones para más o menos los equipos de hoy en día, en plan si pones un número demasiado alto, que diga que el número escogido es demasiado alto y que tienes que poner otro número>> Esto lo ha dicho cuando ha probado a generar un cubo de 800 x 800 x 800, con lo que su ordenador se ha quedado parado y al consultar el rendimiento de la memoria RAM se comprobó que se había llenado.

<<Falta poner que se puede hacer zoom con el ratón>> El contexto de este comentario es que el usuario ha movido la rueda del ratón sin querer y ha comprobado que era posible hacer zoom. Según el usuario estaría bien poner en alguna parte de la aplicación un mensaje para que el usuario sepa que se puede hacer zoom con el ratón.

<<Las pistas despistan. Pista no lleva a la solución. Deberías poner, posibles movimientos>> Cuando el usuario ha utilizado el botón pista, esperaba que la aplicación le mostrara la siguiente casilla a la que se debía mover para salir del laberinto. Sin embargo, ha comprobado sorprendido, que se mostraba más de una casilla, ya que se muestran las posibles casillas a las que se puede mover la bola. El usuario cree que este botón debería renombrarse y pasar a llamarse "posibles movimientos" porque resulta confuso.

<<Al finalizar, ¡qué ponga has terminado! O algo así>> En este caso, el usuario había conseguido salir del laberinto y al terminar se ha molestado porque no ponía que había conseguido salir del laberinto.

Cuando se le ha preguntado por los controles al usuario, solo ha hablado de las flechas y el ratón. Después, se le ha explicado que también se podían hacer otras cosas, como generar otros cubos u otros laberintos, cambiar la vista para ver cómo se imprimiría el laberinto… a lo cual el usuario ha respondido con el comentario <<Ah pero es que eso ya se ve… Es muy intuitivo>>.

Por lo tanto, de este primer usuario de pruebas, destacan las ideas de que hace falta un limitador en las dimensiones y cambiar el botón "pista" a "posibles movimientos". Los controles están dispuestos de forma que el usuario puede saber qué hacer para conseguir lo que el desee en cada momento. Sin embargo, estaría bien informar al usuario de algunos controles (que puede hacer zoom, por ejemplo). La aplicación es entretenida y diferente.

Usuario 2

El segundo usuario es un joven de 20 años con cursando estudios superiores en el ámbito de las ciencias de la salud.

Los resultados para cada una de las preguntas mencionadas anteriormente han sido los siguientes:

1. a. La aplicación genera laberintos en tres dimensiones.
   b. No lo sé.

2. a. Los botones que se ven en el recuadro de la derecha.
   b. Crear un cubo, un laberinto, una bola, cambiar la vista y mover la bola.

3. El usuario ha sido capaz de realizar todas las acciones.

4. Al usuario le ha costado poco realizar las acciones, pero ha tardado considerablemente más tiempo que el usuario anterior. Este usuario también ha tardado bastante en encontrar la solución, pero no ha querido hacer uso de las pistas.

5. Tras pasarle el formulario final al usuario se obtienen las siguientes respuestas para cada una de las preguntas:

   ¿Qué te parece la aplicación?
   Pues muy bien, muy chula. Pero es difícil encontrar la salida del laberinto.

   ¿Qué es lo que más te ha gustado la aplicación? ¿Por qué?
   La apariencia. Cómo se muestran los botones. Y también poder crear laberintos automáticamente, es impresionante. Porque no conozco nada parecido.

   ¿Qué es lo que menos te ha gustado de la aplicación? ¿Por qué?
   Que es complicado ver por donde está la salida del laberinto. Porque hace incómodo jugar con él.

   ¿Crees que falta algún control? ¿Cuál?
   Sí. Poder ver la solución si no puedes encontrar la salida.

   ¿Crees que sobra algún control? ¿Cuál?
   No.

   ¿Cambiarías la visualización de algún elemento? ¿Cuál/cuáles? ¿Cómo?
   Sí. El cubo, para verlo mejor, se podría dejar al usuario bajar o subir un piso y que no se vea ese piso para ver a través.

   ¿Cómo mejorarías la aplicación?
   Cambiando la visualización del cubo, poniendo cómo se debe usar y poniendo un botón “solución” para ver la solución del laberinto.

Trabajo futuro

Gracias a los resultados de las pruebas y el primer análisis que se hizo sobre las herramientas existentes, se pueden observar diversas mejoras que se podrían llevar a cabo sobre nuestro generador de laberintos en 3D.

Basándonos en los usuarios de pruebas, parece necesario añadir información de cómo utilizar algunos de los controles: "que se puede hacer zoom, girar el laberinto…" Se podría añadir al inicio un video tutorial, o simplemente un botón de ayuda que muestre los controles de la herramienta.

También se deberían cambiar algunos de los textos de los botones y limitadores que impidan crear cubos de dimensiones excesivamente grandes.

Una de las propuestas más interesantes de uno de los usuarios ha sido la posibilidad de permitir adentrarse en el laberinto en primera persona. Aunque quizás se aleje más de la herramienta que se ha querido crear, sería una buena forma de mejorarla si se quiere hacer un juego complejo e innovador.

Además de las mejoras que han comentado los usuarios de prueba, también están las mejoras que se pueden llevar a cabo basándonos en otras aplicaciones estudiadas en la fase de estado del arte.

Por una parte, una manera muy sencilla de enriquecer la aplicación, sería añadir características encontradas en otras aplicaciones como la posibilidad de mostrar la solución.

Otra mejora muy fácil de implementar es adaptarlo mejor a un juego de tal manera que se implante un sistema de niveles según los cuales la dificultad del laberinto aumenta.

Con modificaciones más complejas, se podría añadir la posibilidad de crear laberintos con distintas formas. El usuario podría elegir, por ejemplo, si desea crear un laberinto en una esfera, una pirámide…

Conclusión

El resultado final obtenido cumple los objetivos propuestos al inicio del trabajo e implementa alguna mejora, por ejemplo, el botón de pistas. La metodología utilizada para el proyecto ha sido la adecuada puesto que finalmente se ha observado que el resultado es el deseado y el desarrollo se ha llevado a cabo de manera sencilla y sin imprevistos fuera de lo planificado anteriormente. Las pruebas realizadas han sido de gran ayuda tanto para detectar errores como para encontrar mejoras.

Bibliografía

Contenido	Referencia
Qué es WebGL y THREE.js	Chris Botman (5 de febrero de 2016). An introduction to Three.js. https://humaan.com/blog/web-3d-graphics-using-three-js/
Historia THREE.js	Three.js (sin fecha). https://es.wikipedia.org/wiki/Three.js
Código de la librería	Ricardo Cabello (abril de 2010). Three.js. https://github.com/mrdoob/three.js/
Tutorial three.js	Nick Pettit (sin fecha). The Beginner's Guide to three.js. http://blog.teamtreehouse.com/the-beginners-guide-to-three-js
Algoritmos para generación de laberintos	Alvy. (29 de octubre de 2016). Un algoritmo para crear laberintos «interesantes». http://www.microsiervos.com/archivo/ordenadores/algoritmo-laberintos-interesantes.html
STL Exporter	Kevin Lubick (sin fecha). STL Exporter. https://gist.github.com/kjlubick/fb6ba9c51df63ba0951f
FileSaver	Eli Grey (sin fecha). FileSaver. https://github.com/eligrey/FileSaver.js/